

SCHEDULING OF RESOURCE- CONSTRAINED PROJECTS

By
ROBERT KLEIN



Springer Science+Business Media, LLC

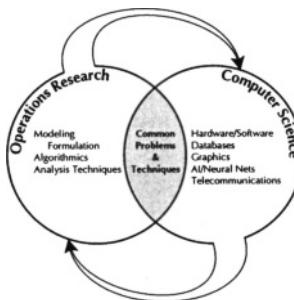
Scheduling of Resource-Constrained Projects

OPERATIONS RESEARCH/COMPUTER SCIENCE INTERFACES SERIES

Series Editors

Professor Ramesh Sharda
Oklahoma State University

Prof. Dr. Stefan Voß
Technische Universität Braunschweig



Other published titles in the series:

Brown, Donald/Scherer, William T.
Intelligent Scheduling Systems

Nash, Stephen G./Sofer, Ariela
*The Impact of Emerging Technologies on Computer Science
and Operations Research*

Barth, Peter
Logic-Based 0-1 Constraint Programming

Jones, Christopher V.
Visualization and Optimization

Barr, Richard S./ Helgason, Richard V./ Kennington, Jeffery L.
*Interfaces in Computer Science and Operations Research: Advances in
Metaheuristics, Optimization, and Stochastic Modeling Technologies*

Ellacott, Stephen W./ Mason, John C./ Anderson, Iain J.
Mathematics of Neural Networks: Models, Algorithms & Applications

Woodruff, David L.
*Advances in Computational and Stochastic Optimization, Logic
Programming, and Heuristic Search*

Scheduling of Resource-Constrained Projects

by

Robert Klein



Springer Science+Business Media, LLC

Library of Congress Cataloging-in-Publication

Klein, Robert.

Scheduling of resource-constrained projects / by Robert Klein.

p.cm. -- (Operations research/ computer science interfaces series ; ORCS 10)

Includes bibliographical references and index.

ISBN 978-1-4613-7093-2 ISBN 978-1-4615-4629-0 (eBook)

DOI 10.1007/978-1-4615-4629-0

1. Production scheduling. I. Title. II. Series.

TS157.5.K554 1999

658.5'3--dc21

99-046684

Copyright © 2000 by Springer Science+Business Media New York

Originally published by Kluwer Academic Publishers in 2000

Softcover reprint of the hardcover 1st edition 2000

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher, Springer-Science+Business Media, LLC.

Printed on acid-free paper.

Contents

Notations.....	XI
Preface	XV

Part I Project Management: Basics and Scheduling Problems

1 The Project Management Process.....	1
1.1 Definition of a Project.....	1
1.2 The Project Life Cycle	2
1.3 Project Conception.....	5
1.3.1 Feasibility Study	7
1.3.2 Economic Analysis.....	8
1.3.3 Risk Analysis.....	10
1.3.4 Project Selection.....	12
1.4 Project Definition	15
1.4.1 Project Specification.....	15
1.4.2 Project Organization.....	16
1.4.3 Process Organization	17
1.4.4 Budgeting.....	19
1.5 Project Planning.....	22
1.5.1 Structuring	22
1.5.2 Scheduling	24
1.5.3 Resource Allocation	24
1.6 Project Execution	26
1.6.1 Reporting, Monitoring, and Control	27
1.6.2 Configuration Management.....	28
1.6.3 Quality Management	29

1.7 Project Termination.....	30
1.7.1 Final Evaluation and Reporting.....	30
1.7.2 Dissolution.....	31
 2 Project Planning and Control.....	 33
2.1 Structuring.....	34
2.1.1 Work Breakdown Structure	34
2.1.2 Activity-on-Node Networks	37
2.1.3 Activity-on-Arc Networks.....	41
2.2 Scheduling.....	43
2.2.1 Critical Path Analysis	43
2.2.2 Slack Time Computations	46
2.2.3 Gantt Charts.....	49
2.3 Resource Allocation	50
2.3.1 Resource Loading.....	50
2.3.2 Resource-Constrained Scheduling.....	52
2.3.3 Time-Constrained Scheduling.....	54
2.4 Control	55
2.4.1 Schedule Control	56
2.4.2 Cost Control.....	60
2.5 Project Management Software	63
2.5.1 Features for Project Conception, Definition, and Termination	64
2.5.2 Features for Project Planning	65
2.5.3 Features for Project Execution	68
2.5.4 General Features	70
 3 Resource-Constrained Scheduling Problems.....	 73
3.1 Notations and Definitions	73
3.2 Basic Models	76
3.2.1 The Resource-Constrained Project Scheduling Problem (RCPSP).....	77
3.2.1.1 Properties of RCPSP	77
3.2.1.2 Formulation 1	79
3.2.1.3 Formulation 2	80
3.2.1.4 Formulation 3	82
3.2.1.5 Formulation 4	84
3.2.1.6 Formulation 5	86
3.2.1.7 Formulation 6	87

3.2.2.1 Properties of GRCPSP.....	91
3.2.2.2 Formulations.....	92
3.2.3 Problem Complexity.....	92
3.3 Extensions of the Basic Models.....	95
3.3.1 Preemption.....	96
3.3.2 Multiple Modes.....	96
3.3.3 Maximum Time Lags	99
3.3.4 State Preserving Jobs	100
3.3.5 Further Extensions.....	102
3.4 Related Project Scheduling Problems	102
3.4.1 The Time-Constrained Project Scheduling Problem	103
3.4.2 The Resource Leveling Problem	104
3.4.3 The Resource Investment Problem.....	105
3.4.4 The Net Present Value Problem	106
3.4.5 The Weighted Tardiness Problem	108
3.4.6 Further Resource-Constrained Project Scheduling Problems	108

Part II Resource-Constrained Project Scheduling: Solution Methods

4 Lower Bound Methods.....	113
4.1 Constructive Lower Bound Methods for RCPSP	114
4.1.1 Simple Bound Arguments	114
4.1.1.1 Critical Path and Capacity Bounds.....	114
4.1.1.2 Bin Packing Bounds.....	117
4.1.1.3 Node Packing Bounds	120
4.1.1.4 Parallel Machine Bounds.....	124
4.1.1.5 Precedence Bounds.....	128
4.1.2 Complex Bound Arguments	129
4.1.2.1 LP-Relaxation with Cutting Planes	130
4.1.2.2 Lagrangean Relaxation.....	132
4.1.2.3 Set Covering Based Approach	134
4.2 Destructive Improvement.....	136
4.2.1 Meta-Strategies for Computing Lower Bounds.....	136
4.2.2 Applying Destructive Improvement to RCPSP	141
4.2.2.1 Reduction Techniques	141
4.2.2.2 Lower Bound Arguments for Contradicting Feasibility.....	147

4.3 Lower Bound Methods for GRCPSP	149
4.3.1 Simple Bound Arguments	150
4.3.1.1 Critical Path and Capacity Bounds.....	152
4.3.1.2 Node Packing Bounds	153
4.3.1.3 Parallel Machine Bounds.....	157
4.3.1.4 Precedence Based Bounds.....	158
4.3.2 Destructive Improvement	159
5 Heuristic Procedures	161
5.1 Types of Schedules	162
5.2 Priority-Rule Based Methods.....	167
5.2.1 Scheduling Schemes	169
5.2.1.1 Serial Scheduling Scheme	169
5.2.1.2 Parallel Scheduling Scheme	171
5.2.1.3 A Critique of the Scheduling Schemes.....	173
5.2.2 Multiple Planning Directions.....	175
5.2.2.1 Backward Planning.....	175
5.2.2.2 Bidirectional Planning.....	178
5.2.3 Priority Rules.....	181
5.2.4 Multi-Pass Priority-Rule Based Heuristics.....	187
5.3 Improvement Methods	190
5.3.1 The Meta-Heuristic Tabu Search.....	191
5.3.1.1 Moves, Neighborhood, and Descent Procedures.....	191
5.3.1.2 Basic Principles of Tabu Search.....	193
5.3.1.3 Extensions of the Basic Approach.....	196
5.3.2 The Tabu Search Procedure RETAPS	198
5.3.2.1 Definition of the Neighborhood	198
5.3.2.2 Tabu Management and Diversification	204
5.3.3 Other Meta-Heuristic Based Procedures for RCPSP.....	208
6 Exact Procedures	213
6.1 Components of Branch and Bound Procedures	214
6.1.1 Branching Schemes	215
6.1.2 Search Strategies.....	216
6.1.3 Bounding Rules	218
6.1.4 Reduction Rules.....	219
6.1.5 Dominance Rules.....	220

6.2 The Branch and Bound Procedure PROGRESS.....	221
6.2.1 The Branching Scheme.....	222
6.2.2 Local Lower Bound Method.....	224
6.2.3 Bounding Rules	226
6.2.4 Reduction and Dominance Rules	228
6.2.4.1 Core Time Rule	228
6.2.4.2 Active Schedule Rules.....	229
6.2.4.3 Supersession Rule.....	231
6.2.4.4 Schedule Storing Rules	232
6.2.5 Example	236
6.3 Scattered Branch and Bound.....	240
6.3.1 Principles of Scattered Branch and Bound	241
6.3.1.1 A Critique of Traditional Branch and Bound.....	241
6.3.1.2 Subdividing the Solution Space into Regions	243
6.3.1.3 Swapping Regions.....	245
6.3.2 SCATTER: Scattered Branch and Bound for GRCPSP	247
6.3.2.1 Outline	247
6.3.2.2 Decomposing the Solution Space.....	248
6.3.2.3 Swapping Regions.....	249
6.3.2.4 Example.....	251
6.4 Existing Procedures	252
6.4.1 Parallel Branching Scheme.....	253
6.4.2 Serial Branching Scheme.....	254
6.4.3 Delaying Alternatives.....	256
6.4.4 Schedule Schemes	258
7 Computational Experiments	261
7.1 Hardware and Software Environment.....	262
7.2 Complexity Measures and Data Sets.....	263
7.2.1 Complexity Measures.....	263
7.2.2 Data Sets for RCPSP	267
7.2.3 Data Sets for GRCPSP	269
7.3 Lower Bound Arguments	274
7.3.1 Simple Bound Arguments	275
7.3.2 Destructive Improvement	278
7.3.3 Influence of the Problem Structure.....	281
7.3.4 Comparison with Complex Bound Arguments	284

7.4 Heuristic Procedures.....	286
7.4.1 Priority-Rule Based Heuristics	286
7.4.1.1 Combinations of Scheduling Schemes and Priority Rules	287
7.4.1.2 Influence of the Problem Structure.....	290
7.4.1.3 Multi-Pass Performance	293
7.4.1.4 Comparison to Proprietary Heuristics of Standard Software	295
7.4.1.5 Results for GRCPSP.....	296
7.4.2 The Tabu Search Procedure RETAPS	300
7.4.2.1 Analysis of GRCPSP Performance	301
7.4.2.2 Comparing RETAPS to Multi-Pass Heuristics.....	304
7.4.2.3 Comparing RETAPS to Other Heuristic Procedures for RCPSP ..	305
7.5 Exact Procedures.....	306
7.5.1 The Branch and Bound Procedure PROGRESS	306
7.5.1.1 Comparing PROGRESS to GDH.....	307
7.5.1.2 Analyzing the Efficiency of PROGRESS.....	309
7.5.2 Scattered Branch and Bound	313
7.5.2.1 Comparing SCATTER to PROGRESS.....	313
7.5.2.2 Comparing SCATTER to Existing RCPSP Procedures	318
7.5.2.3 Comparing SCATTER to RETAPS.....	321
8 Summary and Conclusions	325
References	333
Index	365

Notations

General Notations and Symbols

$x := y$	x is defined by the value of y
$\lceil x \rceil$	smallest integer $\geq x$
$\lfloor x \rfloor$	largest integer $\leq x$
$ s $	absolute value of a number s
$ S $	number of elements in set S (cardinality of S)
∞	an infinite number
$s \in S$	s is element of set S ; s is in the interval S
$S \subseteq Q$	S is subset of Q
$S \subset Q$	S is proper subset of Q
$S \cup Q$	union of the sets S and Q
$O(f(n))$	order of a function $f(n)$
LP	linear programming
LB	lower bound on the value of an objective function
UB	upper bound on the value of an objective function
RCPSP	resource-constrained project scheduling problem
GRCPSP	generalized resource-constrained project scheduling problem
cf.	confer
p.	page
pp.	page and following pages

Notations for Resource-Constrained Project Scheduling

n	number of jobs
J	set of all jobs; $J = \{1, \dots, n\}$
j	index for the jobs; $j = 1, \dots, n$
d_j	duration of job j in periods
rd_j	release date of job j
dd_j	due date of job j
P_j / F_j	set of jobs which immediately precede / follow job j
P_j^* / F_j^*	set of jobs which precede / follow job j
A	set of direct precedence relationships ($= \{(i,j) \mid i, j \in J \text{ and } i \in P_j\}$)
A^*	set of all precedence relationships ($= \{(i,j) \mid i, j \in J \text{ and } i \in P_j^*\}$)
λ_{ij}	start-to-start minimum time lag between two jobs i and j in number of periods
\bar{T}	end of the planning horizon
t	index for periods; $t = 1, \dots, \bar{T}$
CT	completion time of a project
CP	critical path
ES_j	earliest starting time of job j
LS_j	latest starting time of job j
EF_j	earliest finishing time of job j
LF_j	latest finishing time of job j
TSL_j	total slack time of job j ($= LS_j - ES_j$)
α_j	head of job j ($= ES_j$)
ω_j	tail of job j ($= LF_n - LF_j$)
ψ_j	start tail of job j ($= LF_n - LS_j$)
$E(t)$	jobs which are eligible in period t ($= \{j \mid j \in J \text{ and } ES_j + 1 \leq t \leq EF_j\}$)

m	number of renewable resource types
R	set of all renewable resource types; $R = \{1, \dots, m\}$
r	index for renewable resource types; $r = 1, \dots, m$
a_r	constant per period availability of resource type r
a_{rt}	availability of resource type r in period t
a_r^{\max}	maximum availability of resource type r in the periods $t = 1, \dots, \bar{T}$; $(= \max\{a_{rt} \mid t = 1, \dots, \bar{T}\})$
u_{jr}	per period usage of resource type r by job j
IP	collection of all incompatible job pairs
IS	collection of minimal resource incompatible sets
CS	feasible (complete) schedule
PS	feasible partial schedule
$J(PS)$	jobs which are scheduled within a partial schedule PS
$SS_j(PS)$	scheduled starting time for job j in PS ; $j \in J(PS)$
$SF_j(PS)$	scheduled finishing time for job j in PS ; $j \in J(PS)$
$ES_j(PS)$	schedule dependent earliest starting time for job $j \in J - J(PS)$
$EF_j(PS)$	schedule dependent earliest finishing time for job $j \in J - J(PS)$
$A(PS)$	jobs which are available for the partial schedule PS ; $(= \{j \mid j \in J - J(PS) \text{ and } P_j \subseteq J(PS)\})$
$E(PS, t)$	jobs which are eligible for the partial schedule PS at time point t ; $(= \{j \mid j \in A(PS) \text{ and } ES_j(PS) \leq t\})$
SSS	serial scheduling scheme
PSS	parallel scheduling scheme
$DFSB$	depth-first search with complete branching
$DFSL$	depth-first search organized as laser search
$LLBM$	local lower bound method
RS	resource strength

Preface

In the last decades, project management has become a wide-spread instrument which enables industrial and public organizations to efficiently master the challenges of steadily shortening product life cycles, global markets and decreasing profit margins. With projects increasing in size and complexity, it reveals that their planning and control represents one of the most crucial management tasks. In particular, this is true for scheduling which is concerned with establishing execution dates for the sub-activities to be performed in order to complete the project. As soon as the limited availability of resources forces conflicts between concurrent projects or even sub-activities of a single project, this task often can not be accomplished without using the support provided by one of the many commercial project management software packages, such as, e.g., Computer Associates Superproject, Microsoft Project, or Scitor Project Scheduler. However, the results yielded by the included solution procedures are often rather unsatisfactory. Due to this reason, the development of more efficient procedures, which can easily be integrated into the software packages by incorporated programming languages, is of great interest for practitioners as well as scientists working in the field of project management.

The book on hand is subdivided into two parts. In **Part I**, the project management process is described and the management tasks to be accomplished during project planning and control are discussed. This allows for identifying the major scheduling problems arising in the planning process among which the resource-constrained project scheduling problem is the most important. Basically, it consists of assigning execution dates to the sub-activities of a project, such that it is terminated as early as possible without exceeding the availabilities of the resources involved in any period of its execution. After defining this problem, a generalized version is introduced which is considered by most of the commercial project management software packages on hand. Finally, a survey on possible extensions which have been examined in the literature so far is given.

Subsequently, **Part II** deals with efficient computer-based solution procedures for the resource-constrained project scheduling problem and its generalized version. Since both problems are NP-hard, the development of such procedures which yield satisfactory solutions in a reasonable amount of computation time

is very challenging. After giving a survey on the extensive research work which has been performed in this area so far, a number of new and very promising approaches are introduced. This includes heuristic procedures based on priority rules and tabu search as well as well lower bound methods and branch and bound procedures which can be applied for computing optimal solutions. Finally, to examine the effectiveness of the new procedures, the results of comprehensive computational experiments are reported.

In particular, I want to thank Prof. Dr. Wolfgang Domschke who provided me with the possibility to perform this work and who accompanied its development with numerous advices and suggestions for improvements. Furthermore, I am grateful to Prof. Dr. Hartmut Stadtler for reading the manuscript and for supporting this work by many inspiring discussions. Special thanks are due to my current and former colleagues Dipl.-Math. Gabriela Krispin, Dr. Armin Scholl, and Prof. Dr. Stefan Voß for the great collaboration during all the years. Without their critical comments, this book would not exist in the present form.

Finally, I owe a debt of gratitude to my future wife Petra. This book is dedicated to my parents.

Darmstadt, August 1999

Robert Klein

Part I

Project Management: Basics and Scheduling Problems

1 The Project Management Process

During the recent decades, a rapid growth in the use of project management as an instrument which enables industrial and public organizations to realize their objectives took place. Project management has been widely accepted as a powerful tool in order to face the challenges of steadily shortening product life cycles, globalization of markets and decreasing profit margins.

In this chapter, a short introduction into the project management process is given and the corresponding management tasks are described. However, only those tasks are addressed which are immediately concerned with the execution of the project. Other ones which more or less accompany the project management process are excluded. Examples are project bidding as well as claim, contract and procurement management. The description is organized along the project life cycle. For comprehensive introductions into project management we refer to Shtub et al. (1994), Badiru and Pulat (1995), Lewis (1995), Meredith and Mantel (1995), Burghardt (1997), Kerzner (1998), and Cleland (1999).

1.1 Definition of a Project

In the literature concerning project management, a large variety of definitions of the term *project* can be found. In general, they all describe a project as a one-time activity with specific objectives which has to be realized in a certain period of time using a limited number of resources. Analyzing the different definitions of projects more deeply gives a broader perspective and leads to the following typical characteristics (cf., e.g., Shtub et al. (1994, pp. 1), Meredith and Mantel (1995, pp. 7), Spinner (1997, pp. 4)):

- A project represents a one-time activity with a set of well-defined objectives. To achieve these objectives, a number of sub-activities has to be accomplished. The duration of a project is restricted, i.e., with reaching the defined objectives, the project is terminated.

- A project is unique in the sense that it possesses some features which avoid completely reducing its execution to routine. Furthermore, it is complex enough that performing the sub-activities requires careful coordination.
- Terminating a project requires the collaboration of several functional departments of a parent organization or even different organizations. Additionally, a project may interact with other projects carried out simultaneously.
- The resources which are available for executing a project are restricted. This refers to the budget as well as to the availability of human resources and equipment.
- The execution of a project may involve a considerable degree of uncertainty. Principal sources of uncertainty include variations in the performance of resources, inadequate or inaccurate data, and the inability to forecast satisfactorily due to the lack of prior experience.

All these characteristics show that performing a project is a complex task resulting in a need for extensive management involvement.

1.2 The Project Life Cycle

Most projects go through similar phases from their initiation until their completion. These phases build the *project life cycle*. Though the phases may vary in their size and complexity and their names may differ depending on the organization, projects typically include those phases shown in Figure 1.1 (cf., e.g., Angus and Gunderson (1997, pp.9) and Lewis (1997, pp. 7)).

Though not always justified, the terms project life cycle and *product life cycle* are often used synonymously (cf. Munns and Bjeirmi (1996)). In particular, this is true, when the objective of the project is to develop a new product or a new system. While the project life cycle usually terminates with the product being sold or the system entering its operational life, the product or the system one may continue far beyond this point. For example, the life cycle of a system may additionally contain an operation and a maintenance phase as well as a disvestment phase. For detailed discussions on the project and product life cycle see Shtub et al. (1994, chapter 10), Munns and Bjeirmi (1996), Kerzner (1998, section 2.7), and Cleland (1999, pp. 49).

At the beginning of the *conceptual phase*, a project is initiated by an organization identifying the need for its realization or by a request from a customer to perform it. Though in this phase there is only a very fuzzy definition of the problem to be solved, a feasibility study, an economic analysis as well as a risk analysis are executed to decide whether to implement the project or not. If due to restricted resources only a subset of all available projects can be realized, the project has to qualify in a selection process. This process can be based on a variety of performance measures such as expected cost, profitability, risk, or resulting follow-on projects.

In the *definition phase*, the project objectives are established. For this purpose, a project specification of requirements is defined in collaboration with the functional departments involved or with the customer. This statement should also consider possible changes of the project objectives and describe how these can be integrated into the specification during the later phases of the project. Subsequently, major development phases are identified and according milestones, i.e., key events terminating the phases, are defined. Furthermore, the project organization is determined. A suitable form of organization is selected, a project manager is appointed and appropriate personnel is acquired. Finally, guidelines for configuration and quality management have to be established.

The *planning phase* starts with sorting out the structure of the project. Based on the project specification, the work content of the project, i.e., the sub-activities which have to be performed for completing it, is defined. For each sub-activity, the expected duration, the required resources and the cost are estimated. Furthermore, precedence relations between different sub-activities are identified. With these data, a schedule is developed for the project by fixing probable execution dates of sub-activities. Defining the schedule represents a complex task because limitations concerning the budget and the availability of resources have to be considered.

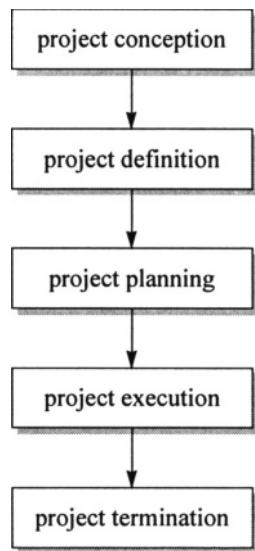


Figure 1.1. Project life cycle

The phases described so far have compromised the first steps in preparing a project for implementation. During the subsequent *execution phase*, the major management task consists of controlling whether the project is performed according to the existent plan. For this purpose, the actual progress, the cost, and the performance of the resources have to be monitored and reported continually. Furthermore, permanent effort has to be made to update original estimates of execution dates, required resources and costs. If deviations from the existing plan are detected, these updates are used for modifying the plan such that the project is put back to course. Besides controlling the execution of the project, quality and configuration management is performed during this phase.

The last phase of the project (*termination phase*) starts when the objectives of the project have been met. By carefully evaluating and reviewing the project, information for improving the management process of subsequent projects is provided. Data on the actual duration and cost of activities as well as the utilization and cost of resources are stored in order to facilitate future planning and control. Finally, the project has to be dissolved, i.e., the participants as well as the equipment have to be reintegrated into the functional departments.

The circumstance that many projects are conducted in a similar way led to the development of *phase models* describing the project life cycle (cf., e.g., Kerzner (1998, section 11.3) and Cleland (1999, chapter 2)). Such models are used to analyze and structure the project management process by identifying the typical management tasks which have to be accomplished in each phase, respectively. Furthermore, they are employed for performing a rough planning of the processing of the project during the definition phase (cf., e.g., Lewis (1995, pp. 46)). A third purpose of phase models is control. At the end of each phase, an audit with the project's *stakeholders*, i.e., the senior management, the project manager, the department managers and the possible customer, can take place assessing the accomplishments of the last phase and getting approval for the subsequent ones.

In some publications concerning project management the authors advocate to organize the management of projects strictly according to phase models (cf., e.g., Kerzner (1998, section 11.3) and Cleland (1999, pp. 291) for details). For different branches of industry, they develop particular guidelines which phases have to be considered and which management tasks have to be accomplished. For example, in systems and software engineering the execution phase is often further subdivided into a design, a production and a test phase (cf., e.g., Boehm

(1981)). Each phase is terminated by a milestone before the subsequent one is started. Overlapping of phases is only allowed in case of exceptions.

However, practice shows that strictly distinguishing different phases has severe drawbacks and rather represents the ideal case. This is mainly due to information and data becoming more precise while performing the project. For example, during the execution phase one of the project's objectives may be subject to changes such that the definition phase is reentered. Furthermore, some management tasks may have to be accomplished repeatedly with progress being made. An economic analysis based on a rough estimate of the project's cost is usually part of the conceptual phase providing input data for the selection process. After the completion of the definition phase, the objectives of the project have been defined and the major sub-activities have been identified such that more evolved estimation methods can be applied. As a result, more versatile phase models have been developed. For software engineering, Boehm (1981) proposed the waterfall model which allows changing the results of a preceding phase in the subsequent one. Later on this model has been extended to the spiral model describing the development of software as an evolutionary process (cf. Boehm (1988)) in which some phases may be entered several times.

In general, a major challenge of project management consists of anticipating decisions and events of subsequent phases. Figure 1.2 gives an overview on the major management tasks of the different phases. A similar presentation is, e.g., contained in Burghardt (1997, section 1.2). Note that the assignment of tasks to phases is not strict, i.e., the same management task may arise in different phases. However, the tasks are assigned to those phases where they are most important from the project manager's point of view, respectively. In the following, we will describe the given tasks as well as corresponding tools in greater detail.

1.3 Project Conception

Usually, a project starts with a proposal either made by some part of the parent organization or by a request of a customer. Before initiating the project, the management of the organization has to decide whether to realize an according project or not. For this purpose, the potential project is commonly analyzed in a preliminary fashion. As a result of this analysis, the project may be discarded because no further effort is warranted, e.g., due to the technological risk or no well-defined market being present. If some promise exists, the project may be

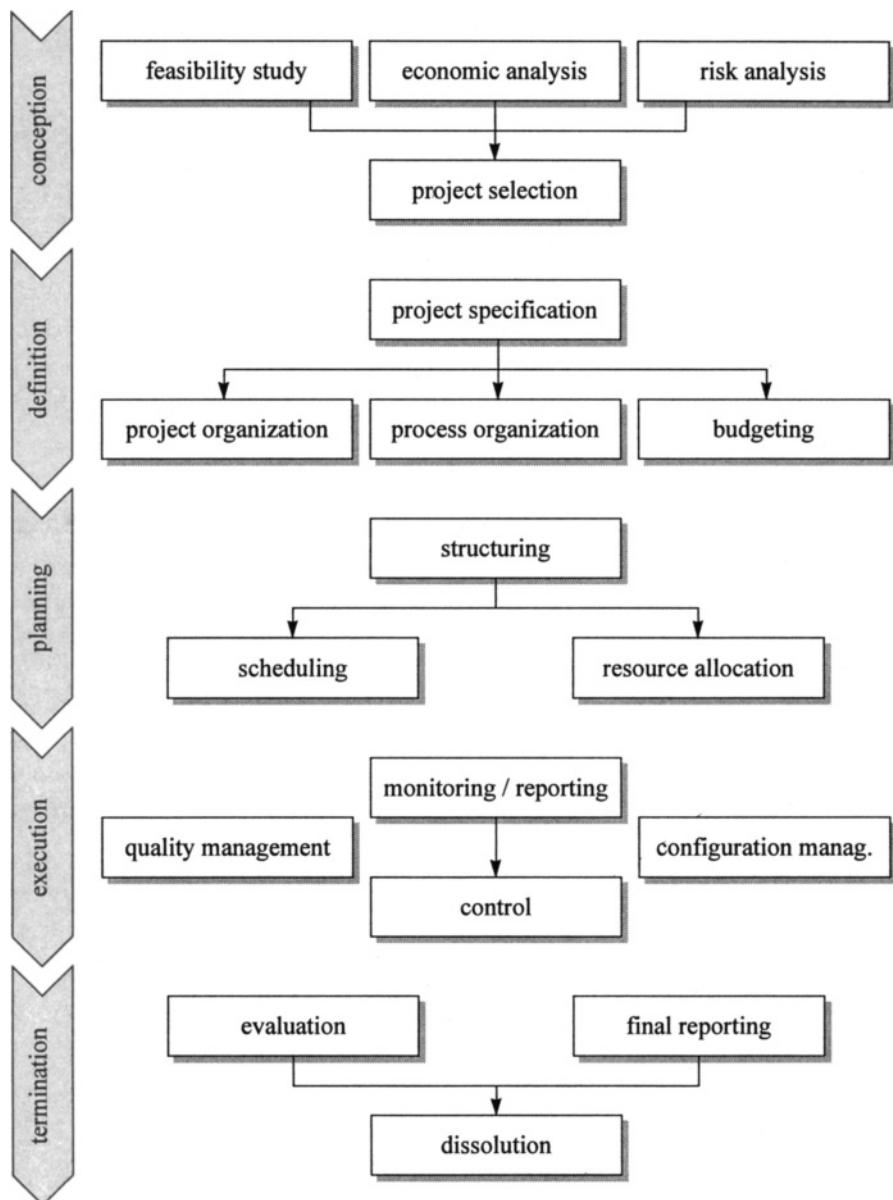


Figure 1.2. Project management process

delayed for some time until the conditions have changed such that the project becomes more attractive. Alternatively, the results may indicate that the project is worth further investigation. In this case, a more in-depth analysis is performed aiming at narrowing the uncertainties associated with the project's costs

and risks. This evaluation usually contains a feasibility study as well as an economic analysis and a risk analysis. If the results indicate that the project should be realized but that it is not superior to other candidate ones, it has to pass through a selection process.

During the conception phase, only a small knowledge about the project and its objectives exists. However, for realizing the studies described below certain assessments have to be made. For example, this may concern the expected cost of the project or the expected development of markets and technologies. To determine these assessments different techniques for *forecasting* exist. In general, these techniques may be categorized into quantitative and qualitative methods. In the main, quantitative methods extrapolate trends from historical data using mathematical methods from statistics such as moving average, exponential smoothing or multiple regression. Qualitative methods may also be based on historical data. However, they rely on subjective judgements of experts. Among the most popular methods of this type are the scenario analysis and the delphi method. For a review on different forecasting methods, we refer to Martino (1983), Wheelwright and Makridakis (1985), and Meredith and Mantel (1995, appendix B). Furthermore, see Pollack-Johnson (1995).

1.3.1 Feasibility Study

Within the feasibility study, it has to be verified whether the project can be realized or not. No predefined guidelines describing how to proceed when performing such a study exist. However, some criteria may be given which can be used as a point of departure for conducting such a study. We restrict to discussing the most important ones. Further criteria may be cultural, social, safety and political feasibility (cf., e.g., Badiru and Pulat (1995, pp. 47), Kerzner (1998, section 11.3)).

First of all, the *technological feasibility* has to be analyzed. It depends on having the corresponding know-how or technology required for performing the project on hand. If this is not the case, it has to be examined whether this know-how or technology can be acquired. The *personnel feasibility* involves ascertaining that sufficient human resources with an adequate qualification for handling this know-how are available within the organization or can be hired. Accordingly, it has to be evaluated whether the *resource feasibility* can be

guaranteed. For example, the required equipment has to be provided and access to facilities like laboratories must be possible.

Financial feasibility is concerned with the question whether the parent organization can raise the funds required for realizing the (sub-)activities of the project within its strategical budget, i.e., without endangering its liquidity. In particular, it has to be taken into account that some sub-activities have to be prefinanced. For example, when realizing a project on request of a customer, cost for equipment, materials and labor are usually payable by the organization continually while processing a sub-activity, whereas corresponding payments of the customer are made after the completion of the sub-activity the earliest.

Finally, the *managerial feasibility* has to be considered. Depending on the size of the project, its organization and implementation may constitute a considerable challenge requiring according experiences and skills of the project managers involved.

1.3.2 Economic Analysis

Each project is performed following the assumption that its realization results in some measurable profit or benefit. Examples are the final payment of the customer who initiated the project, the development of new markets or a better utilization of the organization's resources. Hence, a project always represents an investment which has to be reviewed concerning its profitability. Therefore, already the conception phase should involve an economic analysis. Usually, the selection of appropriate appraisal methods used within such an analysis depends on the type of the project considered.

Most commonly, the economic analysis is based on estimating the *cash flows* occurring during the project or the product life cycle. For this purpose, the expected inflows and outflows as well as their temporal distributions are determined. When a project is performed on the request of a customer, often only the cashflows arising during the project life cycle have to be considered. By the way of contrast, if a new product or system is developed, the cashflows of the whole product life cycle have to be examined due to the long term effect of the project. A typical example deals with selecting components of a new manufacturing system. Often, the short term cost of introducing the system can be reduced by choosing less expensive components. However, this may lead to a

higher probability of failures during the operational life of the system and, hence, to increased maintenance cost and decreased productivity.

Based on the estimated cash flows, various possible *appraisal methods* may be used for evaluating the profitability of the project. Some of these methods consider that money has a time value whereas others do not. The first type of methods are called *dynamic*, the second type *static*. The dynamic methods assume that money can be borrowed or invested at a given interest rate. This interest rate is sometimes also referred to as the discount rate or the minimum attractive rate of return. For a detailed discussion on such methods see, e.g., Herbst (1990, part II), Levary and Seitz (1990, chapter 2), Keown et al. (1994, chapter 10), Kerzner (1998, chapter 14) and Domschke and Scholl (2000, chapter 6). Surveys on further methods focussing on project appraisal are, e.g., contained in Au (1988) and Remer and Nieto (1995).

Among the most frequently used static methods are the payback method and the return on investment method. The *payback method* determines the number of periods necessary to gain a financial return equal to the total investment. A project is considered to be worthwhile if the determined number of periods is below a desired one. The *return on investment method* compares the average annual profit to the total investment. The decision upon realizing a project then usually depends on whether the determined rate excels the minimum attractive rate of return or not.

The dynamic *net present value method* determines the net present value of all inflows and outflows by discounting them by the minimum attractive rate of return. The project is deemed acceptable, if the sum of the net present values of all estimated cashflows over the whole life cycle is positive. The *annual worth method* calculates the average discounted cashflow per period. If this cashflow exceeds the annual payed interest on the initial investment assuming the minimum attractive rate of return, the project is considered to be attractive. The *internal rate-of-return method* is based on the calculation of the interest rate for which the net present value of the project or product is zero. In case of this rate being smaller than the minimum attractive rate of return, the project is discarded. For discussions on the specific strengths and weaknesses of each method, we refer to the literature given above.

In case of the economic analysis examining a complete product life cycle, the cost for realizing the actual project are usually considered as an initial invest-

ment (cash outflow). Thus, only the total cost of the project are of interest. In particular in software engineering, a large number of different methods for estimating the total cost for developing a software product exist (cf., e.g., Gullede et al. (1992) and Schultz (1995)). Among the most wide spread ones are methods based on analogies, factor methods such as the function point method of IBM, and percentage methods (cf., e.g, Chatzoglou and Macaulay (1996) and Burghardt (1997, chapter 3.2)). In the main, the methods differ concerning the information required for their application. For example, the function point method is based on rather detailed informations on the final software product which are usually not available before nearly having finished the definition phase. Therefore, it is usually part of a more in-depth analysis performed when also this phase is finished. Note that all these methods can also be applied in order to estimate the expected cost of performing a sub-activity.

1.3.3 Risk Analysis

Risk analysis encompasses a number of techniques and methods for risk assessment. Insofar possible, its principal intention is to express the possible risks and benefits inherent to the execution of a certain project numerically. In general, three different aspects of risk analysis are important within project management. Most often, only the *economic risk* of a project, i.e., the risk of terminating the project at a loss, is examined. In case of a tight time limit for the project being present, e.g., due to a contract with a customer, the *temporal risk* of finishing the project late can additionally be evaluated. Finally, the *technological risk* of not achieving the objectives of the project has to be considered. While the first two aspects can be addressed using quantitative methods, the last one is usually subject to qualitative forecasting methods such as the scenario analysis and the delphi method. For a recent survey on research relating to project risk management see Williams (1995 a). Introductions are given in Cooper and Chapman (1987), Vose (1996), and Chapman and Ward (1997).

The *economic risk analysis* extends the economic analysis described above by incorporating uncertainty into the input data. Instead of using point estimates of the expected cash flows, probability distributions are determined or estimated. With this input, e.g., the probability distribution of the net present value is calculated either using simulation or analytic methods by taking stochastic sums. Besides yielding probabilistic information on the expected net present value, also its variability as measured by the standard deviation can be determined or

can graphically be represented. A detailed discussion of according methods is, e.g., contained in Herbst (1990, part IV), Levary and Seitz (1990, chapter 3), and van Groenendaal (1998).

A *temporal risk analysis* usually relies on network based methods for project scheduling (cf., e.g., Williams (1995 a)). Such methods are based on determining the sub-activities of a project and identifying precedence relations among them which, e.g., may be due to technological restrictions. Having these data available the project may be depicted graphically in form of a network (cf. Section 1.5.1). Like with the economic risk analysis, point estimates for durations of sub-activities which have to be performed for terminating the project are replaced by probability distributions. However, all these methods require that the project has already been basically structured and therefore can usually not be applied before terminating the definition phase.

The most popular method for considering stochastic durations of sub-activities is called PERT (program evaluation and review technique) (see, e.g., Elmaghraby (1977), Moder et al. (1983), Schwarze (1994), and Shtub et al. (1994, pp. 347) for introductions). Within this method, the optimistic, the pessimistic and the most likely durations of a sub-activity are estimated. Further extensions of PERT, like GERT (graphical evaluation and review technique) even consider the case that sub-activities have to be executed only with a certain probability (cf., e.g., Neumann (1990) for a survey). Recently, also fuzzy methods have been used in order to perform a temporal risk analysis (cf., e.g., Nasution (1994), Shipley et al. (1996)). Approaches which consider uncertainties due to the performance of resources are proposed by Thomasen and Butterfield (1993) and Valadares Tavares et al. (1998).

According to the economic analysis, two basic approaches exist for analyzing the temporal risk based on a project network. A *simulation* approach starts with determining a probability distribution, e.g., a triangle distribution, of the duration for each sub-activity based on the optimistic, the pessimistic, and the most likely estimate (cf., e.g., Ragsdale (1989), Dawson (1995)). In each simulation run, a sample of the duration of each sub-activity is taken. Based on these data, the expected project completion time is calculated (cf. Section 2.2.1). By repeating this process a large number of times, it is possible to yield probabilistic information on the earliest and latest starting and finishing times of each sub-activity as well as for the project completion time. Furthermore, the probability of terminating the project at a given date can be determined by relating the

number of runs where this project completion time has been yielded to the total number of runs.

Within the *analytical* PERT approach, it is assumed that the probability distribution of a sub-activity can be modeled by the beta probability density function. Using approximation formulae, the mean duration and the variance of each sub-activity can easily be calculated after estimating an optimistic, pessimistic and most likely duration (cf., e.g., Williams (1995 b)). Subsequently, the expected project completion time can be computed by referring to the mean durations and by performing a simple critical path analysis as presented in Section 2.2.1. Accordingly, the expected variance of the project completion time may be determined.

Finally, the probability of finishing the project at a certain completion time CT can be evaluated following the central limit theorem (cf., e.g., Burke (1992, section 4.4)). This theorem states that the sum of a set of identically distributed random variables tends to be normally distributed if the number of variables is sufficiently large. Hence, the project duration as a sum of beta-distributed variables is usually assumed to be approximately normally distributed. After calculating the expected completion time and the expected variance as described above, the corresponding normal curve can be constructed. Since the total area under the normal curve is one, the probability can be computed by determining the size of the area on the left-hand side of CT. For a critical discussion on this procedure see Soroush (1994).

In particular, the use of the beta distribution for describing the durations of sub-activities has been heavily criticized in the literature. As a consequence, a large number of proposals which distributions should be assumed instead have been made. Some of the most recent proposals can be found in Bendell et al. (1995), Cox (1995), Lau and Somarajan (1995), Kamburowski (1997), and Lau and Lau (1998).

1.3.4 Project Selection

Project selection can be described as the process of evaluating individual projects and then choosing a subset of them for execution such that the objectives of the parent organization will be achieved. Of course, only those projects are included into the selection process the execution of which does not represent an operating or competitive necessity for the organization. Project selec-

tion can also be applied when different, mutually exclusive alternatives intended to achieve the same objective can be realized.

Basically, there exist two different approaches for project selection (cf. Souder (1984), Shtub et al. (1994, chapter 3), and Meredith and Mantel (1995, section 2.3)). The first one consists of a *prioritizing process* which is based on assessing and ranking each project. For ranking projects, different appraisal methods exist. Most frequently, only the results of the economic analysis are used (cf., e.g., Liberatore and Titus (1983) as well as Watts and Higgins (1987)), i.e., possible projects are, e.g., ranked according to decreasing net present values. However, concentrating on financial data only is sometimes critical. For example, the net present value gives no indication of the scale of effort required to perform the project. That is, two projects may have identical net present values though the associated total cost may differ considerably. A detailed discussion of problems associated with the application of financial appraisal methods can be found in any standard work on financial management (cf., e.g., Keown et al. (1994)).

In order to overcome the disadvantages of concentrating on a single criterion, *scoring models* which refer to multiple criteria for evaluating a project have been proposed (cf., e.g., Burke (1992, pp. 39) and Meredith and Mantel (1995, pp. 55)). In a first step, a checklist containing the criteria and requirements to be considered in the selection process is identified. In the second step, a scoring scale is developed to measure how well a project does with respect to each criterion. Furthermore, the relative importance of each criterion is signified by assigning a specific weight to it. In the last step, each project is examined and its score for each criterion as well as its total weighted score are determined.

A major drawback of using a prioritizing process is that interrelationships between different projects can not be taken into account. For example, two projects may require the same set of resources such that they have to be performed mutually exclusive. Furthermore, realizing a certain subset of projects may result in a total initial investment which is not possible due to budget constraints. A classification of possible interrelationships among projects is given in Gear and Cowie (1980). In order to consider such interrelationships, *integer programming* models are often used for determining an efficient project portfolio.

One of the first models of this type has been introduced by Dean and Nishry (1965). It can be described as follows. A subset of projects has to be selected from $j = 1, \dots, n$ different possible ones. The score or benefit of each project is denoted by s_j . For performing the selected projects, $r = 1, \dots, m$ resource types (labor, budget, etc.) are available with a_r capacity units. The utilization of a resource type r by a project j is assumed to be u_{jr} capacity units. For each project j , a zero-one (binary) variable x_j is introduced which is assigned the value one if the project is selected and the value zero, otherwise. With these notations, the model can be defined as given below.

$$\text{Maximize} \sum_{j=1}^n s_j \cdot x_j \quad (1.1)$$

subject to

$$\sum_{j=1}^n u_{jr} \cdot x_j \leq a_r \quad \text{for } r = 1, \dots, m \quad (1.2)$$

$$x_j \in \{0, 1\} \quad \text{for } j = 1, \dots, n \quad (1.3)$$

The objective function (1.1) aims at maximizing the total score of the projects selected. The constraints (1.2) ensure that the determined project portfolio can be realized with the available resources. Finally, the decision variables x_j are only allowed to have the values one or zero (constraints (1.3)).

The problem described represents a multi-period knapsack problem which also arises in investment planning (cf. Levary and Seitz (1990, chapter 5), and Domschke and Drexl (1998, section 6.6)). In the meantime, a large number of extensions of this basic model have been considered (cf., e.g., Taylor et al. (1982), Fox et al. (1984), Schmidt and Freeland (1992), Schmidt (1993), and Chun et al. (1995)). Coffin and Taylor (1996) address uncertainty by incorporating fuzzy logic into their approach. Efficient solution methods for the multi-period knapsack problem have been developed by, e.g., Gavish and Pirkul (1985) and Drexl (1988). Furthermore, such models have been integrated into decision support systems for project selection (cf., e.g., Güven Iyigün (1993) and Schniederjans and Santhanam (1993)). An approach based on the development of a decision tree is presented in Hess (1993).

1.4 Project Definition

After having selected a project, the definition phase is entered in which the basic conditions of performing the project are determined. Besides defining the objectives of the project more precisely, the project and the process organization are established. Additionally, the budget of the project is developed. The management tasks of this phase have to be accomplished carefully because correcting possible mistakes, e.g., concerning the project specification, in later phases is extremely expensive (cf., e.g., Abdul-Kadir and Price (1995)). The results of the analysis and planning steps performed in this phase are set forth in the *project notebook* which provides a guideline for conducting the project (cf., e.g, Lewis (1995, pp. 35)).

1.4.1 Project Specification

Usually, the project specification is developed step by step with an increasing degree of detail in a top-down approach. At the beginning of the project, only a proposal is available containing the information necessary for the selection process. Subsequently, a *statement of work* or *mission statement* is prepared in co-operation with the client, i.e., either the customer who initiated the project or the part of the organization for which the project is performed. It represents a narrative description of the work required for terminating the project successfully (cf., e.g., Kerzner (1998, section 11.7)). Finally, the objectives of the project are put into a concrete form leading to a detailed project specification.

The *project specification* has to describe which functions the new product or the system has to provide, which interfaces to other systems have to be considered, and which other properties are requested. Its careful preparation is important because it usually represents the basis of further planning as well as of contracts between the organization and the client. Therefore, it has to be comprehensive and free of contradictions. In some industries, this has led to the development of guidelines for the preparation of project specifications. A detailed compilation of possible topics to be considered can, e.g., be found in Burke (1992, pp. 56), Angus and Gunderson (1997, chapter 8), Burghardt (1997, section 2.2), and Kerzner (1998, pp. 587).

1.4.2 Project Organization

As stated earlier, a project represents a one-time activity with restricted duration requiring the collaboration of different functional departments of a parent organization or even of different organizations. In particular, the last characteristic underlines the need for establishing some special kind of organizational structure due to organizations usually being structured functionally (cf. Cleland (1999, chapter 8)). Among the most popular structures are the pure project organization, the influence project organization and the matrix organization (cf., e.g., Shtub et al. (1994, sections 5.1 and 5.2), Badiru and Pulat (1995, chapter 3), Burghardt (1997, section 2.4), and Kerzner (1998, chapter 3)). Each time a project is initiated, one of these structures has to be selected. For a detailed discussion on the advantages and disadvantages of the different structures we refer to Hobbs and Ménard (1993) as well as Meredith and Mantel (1995, chapter 4). Basically, the different project organizational structures can be characterized as follows:

- Within a *pure project organization*, the project organization is separated from the functional one for the duration of the project. The project becomes a self-contained unit with the project manager having full authority and responsibility. During the project, the staff is no longer part of the departments.
- By the way of contrast, no real project manager exists in an *influence project organization*. The staff of the project remains in its respective departments with the department managers having the authority. The progress of the project is controlled by a project coordinator who informs the departments and tries to coordinate their activities. The success of his work crucially depends on the coordinator being supported by the senior management.
- The *matrix organization* is a two-dimensional organizational structure which combines pure project organization and functional organizational structures without separating the staff from its departments. The project manager has the responsibility for the project and possesses the technical authority whereas the disciplinary authority remains with the department managers. As a result, the project manager and the department managers have to work closely together which is assumed to lead to better project results.

Having selected the organizational structure, the project manager has to be appointed and the project team has to be staffed. When choosing a project manager, it is important to consider his leadership abilities, verbal communication skills, and motivation level. A detailed discussion on how to select the project manager and how to manage the project team can, e.g., be found Dinsmore (1993), Thamhain (1993), Lewis (1995, chapter 12), and Cleland (1999, chapters 10, 17, and 20).

1.4.3 Process Organization

For processing a project successfully, it is necessary to subdivide its execution into several phases, e.g., according to key events representing *milestones* (cf., e.g., Shtub et al. (1994, pp. 304)). The end of each phase or the time point of reaching a milestone provides a checkpoint at which the progress of the project can be controlled by its stakeholders, the assumptions of the plan may be verified, a modified plan may be developed, and its specifications may be refined. For this purpose, due dates when to finish a phase or to achieve a milestone are defined. In particular, when performing a project on the request of a customer, prearranging such checkpoints is important and usually becomes part of the contract.

In practice, two different concepts are pursued for structuring a project. The first concept consists of using particular phase models which are refined and adapted to the actual project. The end of each phase serves as a checkpoint. This concept has already been discussed in the context of the project life cycle (cf. Section 1.2) and therefore is not taken up again. As already stated, its major disadvantage is that phases are assumed to be performed sequentially and are not allowed to overlap, i.e., a new phase may not be entered before all sub-activities of the current phase have been terminated.

This has led to the development of the *milestone planning* (cf., e.g., Kerzner (1998, section 11.9), Andersen (1996), and Burghardt (1997, pp. 107)). The checkpoints now are no longer tied to the end of a phase but refer to the termination of particular important sub-activities. Each milestone provides a point of departure for a subsequent set of sub-activities as well as an end point for a preceding set. Furthermore, it represents a result to be achieved, i.e., a condition or a state a project should reach by a certain amount of time. However, it does not define what to do in order to yield the respective result.

Having identified the milestones of a project, these are usually arranged in a *milestone plan* which is a directed graph showing the logical dependencies between the milestones. The milestones are represented as nodes. An arc between a pair of nodes (i,j) indicates a precedence relationship, i.e., milestone j can not be achieved before milestone i . For examples of milestone plans see Andersen (1996) and Burghardt (1997, p. 108). Subsequently, due dates for the different milestones are set forth.

Besides those two concepts, sometimes also the application of *network based planning techniques* is proposed (cf. Section 1.5.2). However, descriptions of such techniques in the literature usually assume that all work packages, i.e., elementary sub-activities, necessary for realizing the project can be determined. In the definition phase, this is not always possible due to proper forecasting being problematic. For example, depending on the results achieved at a certain milestone different work packages may have to be processed in order to proceed to the subsequent one. That is, the execution of additional work packages may become necessary whereas others can eventually be omitted. Therefore, some authors advocate that milestone planning provides the more flexible concept (cf., e.g., Andersen (1996)). Indeed, analyzing the latter two concepts more deeply reveals that they are closely related. In the main, the major difference is due to the level of detail which is considered. In particular, the activity-on-arc based representation of a project is closely related to the milestone plan. In this approach, work packages (jobs) are represented as arcs and events as nodes (cf. Section 2.1.3, pp. 41). Hence, both techniques can favorably be combined and complement each other (cf., e.g., Burghardt (1997, p. 107)). Basically, the processing of the project is structured by milestone planning. As soon as all relevant data are present, e.g., when having reached a milestone, network based planning techniques come into play.

Usually, during the processing of the project, its specification is subject to changes. A typical example deals with the objectives being altered due to necessary modifications concerning the product or the system to be developed. In order to keep track of these changes a configuration management (cf. Section 1.6.2) is conducted according to guidelines which are settled already within this phase. The same is true for quality management (cf. Section 1.6.3).

A further management task in the execution phase consists of controlling the project by the stakeholders at checkpoints and continually by the project man-

ager. In order to accomplish this task, monitoring and reporting is required. For this purpose, also guidelines should be defined (cf. Section 1.6.1).

1.4.4 Budgeting

In general, a *budget* represents the future plans of an organization expressed in financial terms. The budget of any particular project is part of this organizational budget. It usually describes the expected cash inflows and outflows associated with the project as a function of time. When an organization performs several projects simultaneously, their budgets have to be combined centrally in order not to endanger the liquidity of the organization. Restrictions resulting from the organizational budget have already to be considered in the selection process (cf. Section 1.3.4).

A well-defined budget represents an efficient tool for management. Depending on the planning horizon for which they are prepared, strategic, tactical and operational budgets may distinguished. *Strategic* budgets consider a period of several months to several years. They provide an instrument for establishing the long-range objectives of the organization and directing the resource allocation such that these objectives are achievable. By comparing the budget to the actual cash inflows and outflows the performance of the organization can be monitored.

Tactical budgets cover a planning horizon of 12 to 24 months. Usually, they are regularly updated, e.g. every quarter of a year, following a rolling planning horizon approach. They define the expected monthly cash inflows and outflows of the organization's units, i.e., of departments or projects. The major task of tactical budgets is to preserve the organization from financial difficulties. Finally, *operational* budgets refer to certain sub-activities and the resulting cash inflows and outflows and consider periods of several months.

For a project, the corresponding budget sets a framework of constraints in which it has to be performed. For example, it restricts the possibility of allocating additional resources or requires to delay certain sub-activities resulting in cash outflows. Therefore, its careful preparation is of great importance for completing the project successfully. The most simple approach to derive a budget for a project consists of estimating the cash inflows and outflows associated with achieving a milestone (cf. Section 1.3.2). Based on the milestone plan, these cashflows are assigned specific dates from which the budget can be calcu-

lated. Additionally, the outflows caused by overhead cost which can not be assigned to a particular milestone such as quality or configuration management have to be considered. Also, while being in the execution phase, a more evolved budget may be based on the actual schedule by assigning the cashflows to the work packages (jobs) identified in the work breakdown structure (cf. Section 1.5).

However, most often the budget obtained in this way will not match the one desired by the parent organization. Hence, a coordination process becomes necessary to integrate single project budgets and ongoing budgets, such as for production and marketing, into an acceptable organizational one. Within *top-down budgeting*, the strategic budget is defined relying on the judgement and experience of the senior management as well as on historical data. Subsequently, this budget is successively subdivided into tactical and operational budgets by the functional management and the project managers, respectively. By the way of contrast, *bottom-up budgeting* starts with each project manager proposing a budget which allows an efficient and on-time realization of his project. These proposals are integrated by the respective functional managers into a tactical budget which is complemented by the ongoing budget. Finally, the senior management combines all the resulting budgets into a strategic one. Irrespective of which technique is used, there are some shortcomings due to the coordination process flowing in a single direction, either top-down or bottom-up. They can partially be overcome by organizing the process iteratively, i.e., by alternately performing top-down and bottom-up budgeting until convergence takes place. For a more detailed description of the budgeting process as well as a discussion of the advantages and disadvantages of the different coordination strategies we refer to Wildausky (1986), Shtub et al. (1994, section 8.3), as well as Meredith and Mantel (1995, section 7.1). In Meredith and Mantel (1995, section 7.1) also descriptions of further possibilities for designing the coordination strategy such as the planning-programming budgeting system and the zero-base budgeting are contained.

As a result of the coordination process, a project may have to be performed with a budget that requires adapting the process organization, i.e., the milestone plan. The first possibility consists of rearranging the due dates of milestones. If this is not feasible, e.g., due to the project completion time being restricted, the duration of some sub-activities may be adapted by selecting a different technology or by allocating more (or less) resources. This process is usually referred to

as *project crashing* or as project planning under *time-cost trade-offs*. Since this process represents a complex task, a lot of computer-based approaches for accomplishing it have been proposed in the literature.

Generally, the different approaches can be distinguished according to their assumptions concerning the relationship between the duration of a sub-activity and the resulting cost. Most commonly, a normal duration and a crash duration as well as the associated cost are determined. The former equals the duration which is necessary for performing a sub-activity without additional resources such as overtime or improved equipment. By the way of contrast, the crash duration equals the smallest duration possible when the sub-activity is fully expedited. In general, it is assumed that the duration-cost relationship between these two extreme points can be described by a function. If this function is linear, the problem of determining the durations of the sub-activities such that the project is completed on time with minimum cost can be solved by linear programming and adapted network-flow algorithms (cf., Fulkerson (1961), Kelley (1961), Phillips and Dessouky (1977), Hamacher and Tufekci (1984), and Baker (1997)). Non-linear duration-cost relationships are, e.g., examined by Nair et al. (1993), Levner and Nemirovski (1994), and Deckro et al. (1995). However, depending on the resources involved into the execution of a sub-activity (e.g., machines), also discrete duration-cost relationships may be present. Recent approaches as well as exact solution procedures are, e.g., described in Elmaghraby (1993), De et al. (1995) and Demeulemeester et al. (1996). Heuristic algorithms are discussed in Li and Willis (1993) and Skutella (1998). A general discussion on the economical foundation of such relationships is, e.g., contained in Knolmayer and Rückle (1976).

Furthermore, in case of the cash outflows exceeding a certain limit, realizing a budget may require to borrow cash in which case interest cost may incur. Vice versa, the possible profits due to progress payments made by a customer may be invested until they are required again (cf., e.g., Love (1983)). Therefore, in particular in long-term projects with considerable cashflows, it may be appropriate to prepare the budget such that the *net present value* of the project is as large as possible (cf., e.g., Bey et al. (1981) and Kolisch (1997)). Basically, this can be achieved by delaying the milestones associated with cash outflows as long as possible and realizing those ones resulting in inflows as early as possible. Unfortunately, finding the optimal milestone dates and, hence, the optimal budget under this objective is difficult when precedence relationships between

milestones have been considered. As with project crashing, this led to the development of a number of computer-supported planning tools starting with the fundamental works of Russell (1970) and Grinold (1972). More evolved procedures with less restrictive assumptions are presented by Elmaghraby and Herroelen (1990), Herroelen and Gallens (1993), Sepil (1994), Etgar et al. (1996), Kazaz and Sepil (1996) and De Reyck (1998). However, the possibilities of preparing a budget accordingly may considerably be restricted by contractual arrangements (cf. Herroelen et al. (1997) for a survey). An according approach considering such arrangements is, e.g., presented by Dayanand and Padman (1997, 1999). Finally, a lot of research has been performed additionally considering the restricted availability of resources (cf. Section 3.4.4). An approach which integrates crashing aspects and net present value considerations is presented in Sunde and Lichtenberg (1995).

1.5 Project Planning

After having finished the definition phase, the planning phase is entered in which the execution of the project is prepared in detail. However, not the execution of the complete project may always be subject to the actual planning. Instead, only those sub-activities may be considered the processing of which is required to terminate a phase or to achieve a subset of advised milestones. The decision whether to plan the project execution completely or only partially usually depends on its expected duration. Due to detailed data, such as expected resource requirements and availabilities, being required, the chosen planning horizon usually covers a period of several weeks or at most a half of a year. Furthermore, the planning phase may always be reentered when control reveals the necessity of adapting the current plan. In contrast to the management tasks in the preceding phases, project planning is usually performed by the project manager without the involvement of the other project's stakeholders. Since the focus of this book lies on computer supported project planning, the tools helping to accomplish the management tasks of this phase are described separately in Chapter 2.

1.5.1 Structuring

The first major step in the planning process after the definition phase is the development of the *work breakdown structure* (cf. Section 2.1). It helps the

project manager to subdivide the work into small manageable work packages. Most commonly, the work breakdown structure is defined as a multi-level tree representing the sub-activities and work packages to be executed or the product (system) components to be developed (cf., e.g., Burke (1992, chapter 4), and Kerzner (1998, section 11.10)). The root node of the tree represents the project itself. On the first level, the major sub-activities are identified. On subsequent levels, each sub-activity is broken into smaller ones again (cf., e.g., Figure 2.1, p. 36). This process is continued until a further subdividing of any sub-activity is not useful. This is, e.g., the case when the sub-activity corresponds to a *work package* which can be performed by a single member of the project team. In the following, to distinguish such work packages from those on higher levels of the tree, they are also referred to as *jobs*.

Subsequently, *precedence relationships* between the jobs are identified. They may be determined on the basis of technological, procedural or imposed constraints. For example, when moving a large piece of equipment it may be necessary to disassemble it for the transport. Usually, the jobs and the precedence relationships are represented in form of a network diagram consisting of nodes as well as arrows connecting the nodes. The two most popular techniques for drawing such diagrams are the *activity-on-node* and the *activity-on-arc* representation. In the first approach, the nodes are used to depict jobs and the arcs define the precedence relationships. In the latter one, arrows correspond to jobs, while nodes represent starting and finishing events of jobs. Both techniques are discussed in detail in the Sections 2.1.2 and 2.1.3. Finally, for each job, the duration, the cost and the resource requirements are estimated. The estimates may be based on historical data as well as on quantitative or qualitative forecasting methods.

According to the work breakdown structure, an *organizational breakdown structure* can be defined. It corresponds to a tree depicting the organizational units participating in the project. For example, the project team may consist of a team of engineers, a team of technicians, a team of clerks and the like. The corresponding teams may be managed by a project engineer, a manufacturing engineer, and a project controller, respectively. The leafs of the tree usually represent single team members.

Finally, a *linear responsibility chart* is prepared which integrates the organizational breakdown structure and the work breakdown structure by showing which organizational units are responsible for a job. The responsibility may,

e.g., concern the approval of the jobs, their execution or their control. Usually, the linear responsibility chart is prepared in form of a table. The columns correspond to the organizational units whereas the rows represent the jobs to be performed. Each cell of the table denotes the relationship between a unit and a job. The kind of responsibility is defined within the cell. For the preparation of the organizational breakdown structure and the linear responsibility chart see, e.g., Burke (1992, chapter 4), Shtub et al. (1994, section 5.2.3), and Meredith and Mantel (1995, section 5.4).

1.5.2 Scheduling

In general, the scheduling of projects is concerned with the development of timetables, i.e., the establishment of dates during which the jobs required to complete the project will be executed. The most widely used scheduling approaches are based on forming a network that graphically depicts the relationships between the jobs of the project as described above.

Based on the network representing the project, scheduling is performed by applying *forward pass* and *backward pass* procedures (cf. Section 2.2.1). This results in earliest and latest starting and finishing times for the jobs. Furthermore, the amount of float or slack time associated with each job, i.e., the amount of time its duration may be prolonged or delayed without increasing the end of the project, can be calculated (cf. Section 2.2.2). Those jobs having no slack are considered to be critical and have to be payed particular attention by the project manager. Historically, the corresponding computations are often referred to as CPM (critical path method) analysis.

Having available such information, the project manager now can set up a schedule by assigning starting times to all jobs. In order to communicate the resulting schedule, often a *Gantt chart* which represents a special type of a bar chart is used. On the vertical axis, the jobs to be performed are enumerated. Their starting times and durations are depicted on the horizontal axis (cf. Section 2.2.3).

1.5.3 Resource Allocation

The discussion so far assumed that the only constraints on the schedule have been imposed by the precedence constraints. However, in most projects there are additional constraints which have to be considered. Typical examples deal

with the restricted availability of resources and the budget constraints. Ignoring such constraints can lead to unreliable schedules and the project running overdue as well as over budget. The impact of limited resource availability on scheduling is, e.g., examined in Willis (1985), Li and Willis (1991), Woodworth (1993), Just and Murphy (1994), and MacLeod and Petersen (1996). Drexel et al. (1998 b) discuss the effects of resource constraints on the coordination process in project management.

In general, three different classes of resource types may be distinguished as follows. *Renewable* resource types are available in limited quantities in every period (e.g., labor, equipment) of the planning horizon. For a second class of resource types, e.g., such as materials, the total availability may be restricted over the duration of the complete project. These are *non-renewable* (depletable) resource types. Finally, *doubly constrained* resource types are limited both in their per period availability as well as in their total availability. The budget available for a project is a typical example of such a resource type.

Depending on a given schedule for a project to be performed, the *resource loading profile* of the renewable and doubly constrained resource types can be determined for every period. If for any resource type the amount of resources required exceeds the limit in one or more periods, a *resource conflict* occurs and the given schedule may not be realized. In this case, one of the following techniques may be applied in order to obtain a resource feasible schedule:

- At first, the starting times of jobs having a slack time may be modified such that the resource conflicts are resolved.
- Sometimes, the duration of jobs may be extended by processing them using fewer resources. Following this idea, it can be examined, if according reductions of the resource usages which are possible without exceeding the slack times of jobs are sufficient to remove the resource conflicts. Alternatively, some jobs may also be performed using other resource types which are less utilized.
- A further possibility may consist in splitting some jobs into sub-jobs without significantly altering the precedence constraints. Of course, in this case setup times caused by the preemption of jobs have to be considered.
- The availability of resource types involved into resource conflicts may be increased in the affected periods, e.g., by approving overtime or by lending additional machines. However, the corresponding cost must not exceed the

available budget. Hence, a resource feasible schedule observing the precedence constraints and the given project completion time has to be found such that the additional resources necessary for resolving the resource conflicts can be provided at minimum cost. The problem of determining such a schedule is referred to as *time-constrained* project scheduling (cf., e.g., Harhalakis (1989) and Meredith and Mantel (1995, p. 397)).

- Finally, if all these techniques fail in finding a resource feasible schedule for a given budget, the project completion time has to be increased. In this case, a schedule has to be determined such that the project is completed as early as possible without violating the precedence constraints or causing a resource conflict. The problem of finding such a schedule is called *resource-constrained* project scheduling problem (cf., e.g., Harhalakis (1989) and Meredith and Mantel (1995, p. 397)).

Applying any of these techniques is a complex task. Therefore, it is usually supported by according decision support systems or project management software packages (cf. Section 2.5).

Having determined a resource feasible schedule, *resource leveling* may be performed (cf., e.g., Ashok Kumar and Rajendran (1989), Easa (1989), as well as Seibert and Evans (1991)). It can be defined as the rearranging of jobs within their slack times such that fluctuations in the resource loading profiles are minimized. The rationale behind this effort is that using resources more steadily tends to reduce resource costs. For human resources, it is assumed that cost increase with the need to hire, fire and train personnel. Considering materials, fluctuating demand rates usually result in an increased storage requirement. Furthermore, more effort is necessary for material planning and control. Again, appropriate algorithms exist to accomplish this task (cf. Section 3.4.2).

1.6 Project Execution

In the former sections, planning has been described as a fundamental component of project management. However, even accomplishing the corresponding management tasks carefully does not guarantee that the project is terminated successfully. This is due to planning being based on assessments anticipating future developments. For example, a project plan relies on the estimation of such factors as job durations, resource availabilities, and cost, each of which may be subject to uncertainty. Thus, a major task during the execution of a

project consists of continually monitoring and controlling its progress and, if necessary, adapting the original plan to the actual conditions. This process is usually referred to as project control. Besides controlling that the project is performed in time and on budget, it has also be assured that the resulting product or system matches the project specification set forth in the definition phase. This task is accomplished by configuration management and quality management which accompany the project execution phase.

1.6.1 Reporting, Monitoring, and Control

In the monitoring and control process, performance data are considered describing the progress of the project quantitatively. In particular, these are durations of jobs, usages of resources as well as cost.

Most often, reporting, monitoring, and control on the one side and adapting the current plan on the other side are described as a revolving process. At the end of the planning phase, a *baseline plan* consisting of a schedule and a budget has been developed. During the execution of the project, performance data describing its progress are reported. These data are prepared in the monitoring process such that deviations from the baseline plan can be detected. The control process identifies deviations that require adapting this plan. By incorporating these adoptions, a modified baseline plan is developed and the project execution is continued until the need for adoptions arises again.

As indicated, *reporting* and *monitoring* are concerned with collecting and preparing performance data which is required for control purposes. Therefore, guidelines which performance data to evaluate and how they should be measured have to be established in the definition phase. For example, performance data can be collected periodically, e.g., every week or month or continually each time a job is completed or a milestone is achieved. A comprehensive discussion of different approaches can, e.g., be found in Meredith and Mantel (1995, section 10.2) and Burghardt (1997, sections 4.1 to 4.3). In general, the more detailed the performance data is reported the earlier deviations from the baseline plan can be detected. If network planning techniques have been used to derive the schedule and the budget, reporting often refers to the jobs which have already been completed or which are still in progress. For the first group of jobs, the durations and the actual cost are measured, whereas for the active jobs the remaining durations and expected cost are additionally estimated. Note

that collecting performance data has become much more easier due to the possibilities which are offered by electronic communication tools such as, e.g., e-mail. With these tools the collection process can be executed nearly automatically (cf. Schönert (1998) and Section 2.5.3).

Control can be conducted with respect to the baseline schedule or the baseline budget. *Schedule control* in its most simple form is based on comparing the baseline schedule as, e.g., depicted by a Gantt chart with the actual performance (cf. Section 2.2.3). In this case, reporting is usually performed periodically. Simple *cost control* relates the actual, accumulated cost for the jobs completed so far to the budgeted cost. However, determining the cost of a job is usually more complex than evaluating its duration. A typical example deals with labor cost of a job. In conventional cost control systems often only the total cost of labor in a certain period are considered. Then, a problem consists of dividing this cost upon the jobs processed so far. Therefore, often a special project cost monitoring and control system has to be introduced which orients at the organizational breakdown structure and the work breakdown structure (cf., e.g., Ellis (1993), Shtub (1994, section 11.2), and Burghardt (1997, section 3.1.3)).

Restricting to such simple forms of schedule and cost control may have severe disadvantages (cf., e.g., Burke (1992, section 12.1)). Consider the example of a project running late. As a result only a subset of all sub-activities which have been scheduled to be accomplished in a given period have been executed. Hence, the actual cost may be below the budgeted one resulting in the wrong impression that the project runs on budget. This shows, that the actual progress and the actual cost must be related in order to represent an sufficient measure for cost control. An according instrument is provided by the *earned value analysis* which is described in Section 2.4.2.

1.6.2 Configuration Management

While developing a product or a system, the initial specifications set forth in the definition phase may be subject to changes for a variety of reasons. Configuration management is concerned with managing such changes as well as with providing instruments for integrating them into the specifications. It is essential for performing quality management which is described in the following section. For a more detailed description on configuration management we refer to, e.g.,

Eggerman (1990), Shtub (1994, section 6.5) as well as Kerzner (1998, section 11.27).

The configuration management process starts with the *configuration identification* which is part of the project definition phase. Within this step, configuration items are selected. These may be items of the product or the system which, e.g., are technically complex or possess a large number of interfaces to other items. In order to be able to identify such items, a numerical coding system is adopted and each item is assigned a specific number.

During the execution phase, *configuration change control* is performed. In case that the need for a change arises, a change request is initiated. After having collected all relevant data from the organizational units involved, the request is evaluated by a team of experts which decides whether to reject or to approve it. When making this decision the effects on the budget and the schedule as well as possible risks have to be considered. Finally, approved changes have to be integrated into the specifications. This is achieved by preparing and distributing a change approval form among all organizational units affected. If a certain amount of changes has been accepted, *configuration status accounting* may become necessary. By registering all changes approved so far, the current configuration is identified and a new baseline specification is developed.

1.6.3 Quality Management

In industrial projects, the objective often consists of developing a new product or system. In this case, assuring the quality of the obtained result becomes an important task of project management. Basically, the quality of a product or a system can be described as the degree of meeting the specifications set forth in the project definition phase. More generally, the quality depends on the purpose and requirements of the new product or the system. Typical examples deal with reliability, ease of use, and ease of maintenance (cf., e.g., Mendelsohn (1993)).

The quality management process may roughly be subdivided into quality planning and control. Within *quality planning*, a quality assurance plan is developed which identifies measures for judging the quality of a product or system and defines respective standard values (cf., e.g., Shtub et al. (1994, section 6.7.5)). During the execution of the project, *quality control* is performed by applying appropriate review methods at predefined checkpoints to verify whether these values are achieved. If this is not the case, corresponding reactions have to be

initiated in order to put the project back to course. For a survey on review methods we refer to, e.g., Hand and Plowman (1992) and Burghardt (1997, section 4.4).

Furthermore, the quality management process itself has to be planned and controlled. For example, the review methods have to be selected and their proper application has to be supervised. The latter is done by quality audits. In this context, the ISO 9000 standard has been established which provides guidelines for designing the quality management process (cf., e.g., Hall (1995)). If an organization implements its quality management process accordingly, it may achieve an ISO 9000 certification. Since ISO 9000 has been widely accepted, being certified often represents a prerequisite when bidding for a project.

1.7 Project Termination

When the technological objectives of a project have successfully been achieved, the project is usually terminated performing the following steps. In order to facilitate the management of subsequent projects, a project evaluation is conducted and data for forecasting purposes is collected by reporting. Finally, the project is dissolved. However, a project may also be terminated prematurely. Typical examples are budget overruns, technical difficulties or changing market conditions. In general, deciding whether to continue with a project or not is a complex managerial tasks which has been addressed for the first time by Buell (1967). For an extensive discussion on this topic see, e.g., Balachandra and Raelin (1980), Bard et al. (1988), and Meredith and Mantel (1995, section 13.2). Furthermore see Cleland (1999, chapter 14).

1.7.1 Final Evaluation and Reporting

In order to learn from the experiences made throughout the execution of the project, a final evaluation is performed before its dissolution. Often this final evaluation is conducted as an audit in which all organizational units involved in the project participate. The according results are set forth in a final report.

Most commonly, such a final report is structured as follows. At the beginning, the initial statement of work is given. Subsequently, the developed plans such as the schedule and the budget are described. These are compared with the ones realized. The differences are analyzed qualitatively as well as quantitatively

and possible explanations are examined. Furthermore, if an experience data base exists the corresponding results are stored. For example, a cost data base may be a valuable instrument when estimating the probable costs of future projects.

Based on this analysis, the planning tools and processes which have been used are evaluated. If eventual shortcomings can be identified these are cited. Furthermore, planning tools and processes which have been developed during the project are described when there is promise that they can successfully be applied in future projects. The same is true for possible modifications of existing ones. The last part of the report is devoted to measuring the performance of resources. When subcontractors have participated in the project, it is stated whether they should be considered for future projects or not. Finally, the report documents the performance of the project's personnel.

1.7.2 Dissolution

In general, several types of dissolving a project can be distinguished (see, e.g., Meredith and Mantel (1995, section 13.1)). Termination by *extinction* occurs when the project is stopped due to having achieved the objectives or due to having failed. In both cases, the project team members have to be reassigned to new activities and the equipment has to be disbursed. This type is closely related to termination by *integration* which in particular occurs, if projects are performed within a parent organization. Having established a matrix organization, most team members are still affiliated to some organizational unit. Hence, after terminating the project, they are simply reintegrated in their corresponding units. The same is true for the equipment. Finally, termination by *addition* is present when the project team becomes a new organizational unit of the parent organization. Then, the equipment is transferred to the new unit.

2 Project Planning and Control

With projects growing in complexity and size, their planning and control becomes an increasingly important project management function. Elementary methods and tools for accomplishing this task are described in this chapter. Basically, project planning consists of developing a schedule and a budget for the execution of the project. Most commonly, this is done by applying network based planning methods. Due to their importance, a large number of publications describing and discussing such methods are available. Among the most popular approaches are CPM (critical path method) and PDM (precedence diagramming method). Comprehensive introductions into these methods can be found in Küpper et al. (1975), Elmaghraby (1977), Wiest and Levy (1977), Moder et al. (1983), Shtub et al. (1994), Badiru and Pulat (1995), and Domschke and Drexl (1998, chapter 5).

As indicated at the beginning of Section 1.5, project planning usually covers a planning horizon from several weeks up to half a year. Due to this rather short planning horizon in comparison to the duration of mid-size and large projects, the methods presented in this chapter assume that the data provided for planning such as job durations are deterministic. Indeed, practice shows that experienced project managers are able to estimate the required data sufficiently accurate. Therefore, e.g., most of the project management software packages exclude methods like PERT which consider stochastic data. Furthermore, such methods are primarily designed for analyzing purposes (cf. Section 1.3.3).

During the execution of a project, control is performed by comparing its planned performance to its actual progress. Also for this management task a number of established methods exist among which the earned value analysis is most popular detecting both deviations from the planned schedule as well as the budget. In order to allow applying the presented planning and control methods also for large projects, a number of commercial software packages is available containing methods for analyzing, visualizing and controlling projects. The major features of such software packages are reviewed at the end of this chapter.

2.1 Structuring

Structuring the project constitutes the first major step of project planning after having finished the definition phase. Sub-activities and jobs to be accomplished for successfully completing the project are identified and related to each other by developing a work breakdown structure and a network representation of the project. Furthermore, responsibility relationships between sub-activities and organizational units participating in the project are determined. For this purpose, the development of an organizational breakdown structure is useful.

2.1.1 Work Breakdown Structure

In order to allow for successfully planning and controlling a project, it has to be structured by decomposing it into manageable sub-activities of reasonable size and complexity which can be accomplished separately. For each sub-activity, a specification describing its objectives has to be provided. Furthermore, to define a schedule, a budget and to allocate resources, it has to be characterized concerning its duration, cost, and work content. Most important, to ease control, a single member of the project team or a respective group has to be identified who is responsible for its execution. For this purpose, a work breakdown structure is developed. Figure 2.1 shows a simple example for developing a new bicycle. For detailed descriptions as well as for guidelines concerning the preparation of a work breakdown structure, we refer to, e.g., Burke (1992, chapter 4), Hubbard (1993), Kerzner (1995, section 11.10), Lewis (1995, chapter 4), and Burghardt (1997, section 3.1.2).

As a result of milestone planning (cf. Section 1.4.3) taking place in the definition phase, major sub-activities the execution of which is necessary for completing the project have been identified. Some of them may require a considerable amount of time, resources and cost, whereas others may be completed rather fast without presenting technological difficulties. In particular for the first class of sub-activities, it is therefore appropriate to break them down into smaller ones, the handling of which is less complex. Usually, this decomposition is done within a top-down approach. Those sub-activities which are not further subdivided represent *work packages* or *jobs*. In the following, we use the latter term.

However, to avoid getting lost in details and thereby increasing the complexity of planning and control, the process of recursively decomposing a sub-activity into smaller ones is not continued arbitrarily. Most often, criteria like the following ones are defined which indicate that a sub-activity has been subdivided sufficiently and, hence, represents a reasonable work package or job. In order to facilitate collecting the required data, the duration of a job may be restricted to a certain limit. The same may be true for the maximum cost and requirements of resources. Furthermore, to ease control, it must be possible to assign the responsibility of executing a job to a single member of the project team or a defined group of members. Finally, the jobs must clearly be distinguishable from each other concerning their work content such that accomplishing the same piece of work several times is avoided.

Having finished this decomposition process, the sub-activities and jobs determined can be arranged in a form of a tree with its root node corresponding to the complete project. The major sub-activities are depicted on the first level. On subsequent levels, each sub-activity is decomposed again. The leafs of the tree correspond to the identified jobs. The resultant figure is called *work-breakdown structure* (cf. Figure 2.1).

Decomposing sub-activities can be performed following three basic approaches (cf., e.g., Burghardt (1997, section 3.1.2)). Within a *function-oriented* approach, a sub-activity is subdivided according to the development or organizational functions which have to be provided for performing it. In the example of Figure 2.1, this leads to the sub-activities "Construction", "Testing", and "Manufacturing" on the first level of the work breakdown structure. This principle is also applied for decomposing the sub-activity "Manufacturing". Following a *product-oriented* approach, a sub-activity is broken into smaller ones referring to the technological structure of the product or the system to be developed. Considering the example, the sub-activity "Construction" is split accordingly by defining the major technological components of a bicycle. Finally, a *process-oriented* approach may be realized. When choosing this approach, a sub-activity is decomposed according to its processing. However, this approach requires the corresponding process to be structured strictly sequentially. For example, this is the case for the sub-activity "Testing". Before the prototype has been finished, no laboratory experiments can be executed. This again consti-

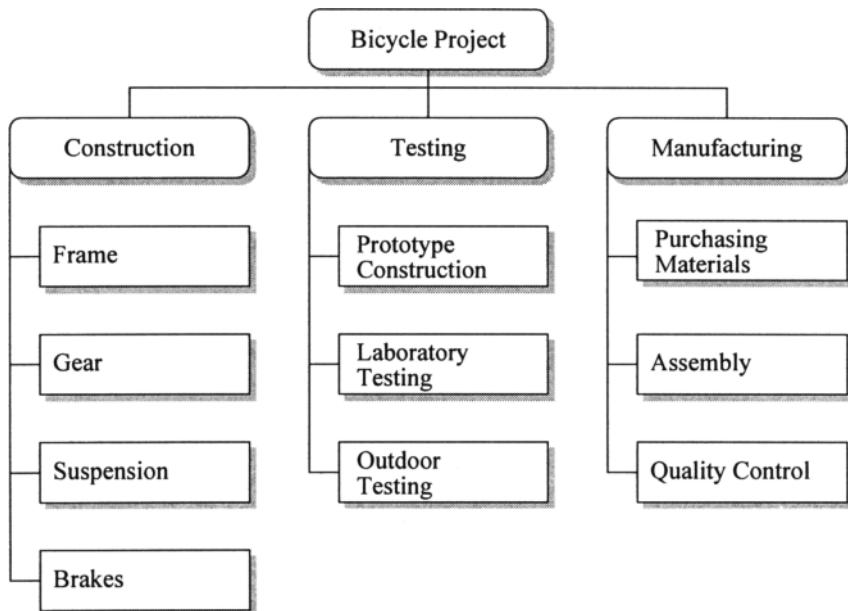


Figure 2.1. Work breakdown structure for bicycle project

tutes the prerequisite for examining the performance of the bicycle under outdoor conditions.

To summarize, the work breakdown structure basically describes the process by which the project is executed. Additionally, for yielding a comprehensive description of the project's work content, the following topics have to be addressed in a separate description of each job (cf., e.g., Shtub et al. (1994, section 5.4) and Kerzner (1998, section 11.7)). First of all, a *statement of work* has to be set forth defining the objective which has to be achieved by performing the job. In this context, to judge its successful completion, according performance measures have to be established. In the next step, the amount of labor, equipment, and materials required for executing the job have to be determined. Based on these data, the expected duration of the job can be estimated. Finally, a budget for executing the job can be developed.

In particular, the satisfactory estimation of job durations constitutes a challenging task which can be accomplished by applying qualitative and quantitative forecasting methods. A detailed discussion of this topic can, e.g., be found in Burke (1992, chapter 9) and Shtub et al. (1994, section 7.2). Additionally, if

some jobs are performed repeatedly during the project, learning effects may occur which have to be considered (cf. Badiru and Pulat (1995, section 5.10)).

Remark 2.1. Note that commonly the *job durations* are all expressed referring to the same measure of time, i.e., in the number of days, hours, and the like. Most often, this time measure is chosen sufficiently small, such that job durations can be assumed to be integral. To ease the following discussion, a corresponding time unit is called *period*. The begin and the end of each period is defined by a time point, respectively.

2.1.2 Activity-on-Node Networks¹

After having developed the work breakdown structure, all relevant jobs for performing a project have been identified. However, the jobs can usually not be processed in arbitrary order such that precedence relations among pairs of jobs may have to be considered. Basically, three different types of precedence relationships may be distinguished (cf., e.g., Badiru and Pulat (1995, pp. 107)). *Technological* precedence relationships are due to technological interdependencies between different jobs. In the above example of developing a bicycle, selecting the suspension system is not possible before having designed the frame. *Procedural* precedence relationships are caused by guidelines defining how certain processes have to be performed. For example, due to safety reasons, it may not be convenient to execute the outdoor tests before the laboratory tests have been completed. Finally, *imposed* precedence relationships can be present. This is, e.g., the case if one job has to be executed before another because concurrent processing is not possible due to restricted resource availability or budget constraints.

Irrespective of the reason for introducing a precedence relationship between a pair of jobs, the so called "*finish-to-start*" precedence relationship is most commonly used in project scheduling. For example, it is assumed to be present when performing a network analysis applying the CPM or PERT technique (cf. Section 2.2.1). It requires that a job can only start after having completed all those job which have to be processed in advance because of existing prece-

1. Note that the term "activity-on-node network" has been established historically. As stated earlier, in this book the term job is given preference to the term activity when describing low-level work packages which have to be executed. Thus, also the term "job-on-node network" could be used.

dence relationships. In the following, we concentrate on this type of precedence relationship. More evolved types are discussed at the end of the section. Due to the large number of precedence relationships usually present within a project, it is difficult to rely on describing them in a verbal manner only. Therefore, to convey their effects, they are commonly depicted graphically.

In the following, some of the according techniques are presented in greater detail. In order to ease their presentation, the following notations are introduced (cf. Section 3.1). Realizing a project requires performing a set of jobs $J = \{1, \dots, n\}$ defined by the work breakdown structure. Each job can be denoted by a unique number $j \in J$, and the corresponding duration in periods is given by d_j . A job i which must be completed before a job j can be performed is called a *predecessor* of i , while j is a *follower* or *successor* of i . A precedence relationship between a pair of jobs (i, j) is called *direct* or *immediate* if no follower of job i is also a predecessor of j . The immediate predecessors of job j form the set P_j . Accordingly, the set F_j contains all immediate followers of a job j .

When graphically depicting the precedence relations, it is convenient to define a unique *dummy start* and *end job* which identify the milestones of starting and terminating the project. For the sake of simplicity, the jobs 1 and n are defined to represent these jobs. Furthermore, their durations are assumed to be $d_1 = d_n = 0$.

Table 2.1. Data of example project

job j	predecessors P_j	durations d_j
1	–	0
2	1	3
3	1	5
4	1	1
5	2	3
6	3, 4	2
7	3, 5	4
8	6	5
9	6	6
10	6	4
11	7, 8	4
12	9, 10, 11	0

Table 2.1 contains all relevant data of an example project with $n = 12$ jobs. The jobs 2, 3, and 4 only have the single predecessor job 1, i.e., they can be executed immediately after starting the project. Job 6 has two predecessors and, hence, its execution can not begin before both, job 3 and job 4 are terminated. Due to job 3 preceding job 6 and indirectly job 8 (a follower of job 6), job 3 is also a predecessor of job 8. That is, an according precedence relationship is implicitly defined and need not be introduced. In general, such *transitive* precedence relationships are *redundant* and can be omitted.

Having these data available, a corresponding *activity-on-node network*, which is a directed graph, can be constructed. As stated earlier, the nodes of the graph represent the jobs while the arcs depict the precedence relations. Figure 2.2 shows the network which is obtained for our example. The job numbers and durations are given in the left-hand and right-hand node entry, respectively. In case of considering only "finish-to-start" relationships, the graph has to be acyclic. Otherwise, in case of detecting a cycle while drawing the network, a contradiction in the precedence relationships is present.

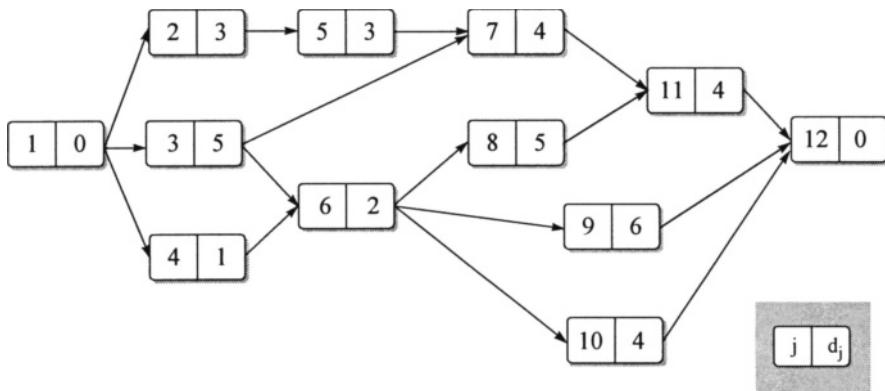


Figure 2.2. Activity-on-node representation of the example project

In general, there are no binding rules for constructing an activity-on-node network. For example, there exists no convention for drawing the nodes. In particular, when using a project management software package, the nodes often contain additional information, such as the names of the respective jobs, their scheduled starting and finishing times as well as cost or resource requirements. However, it is common style that only a single start and end node are drawn. That is, all other nodes have at least one predecessor and successor, respectively.

As indicated, precedence relationships may be further specified in order to provide extended modeling capabilities, e.g., allowing jobs related by precedence to be performed partially in parallel instead of in series. For this purpose, the *precedence diagramming method* (PDM) has been developed which introduces new types of precedence relationships (cf., e.g., Crandall (1973), Wiest and Levy (1977), and Moder et al. (1983)). First of all, *minimum time lags* between jobs are introduced. A "finish-to-start" time lag λ_{ij}^{FS} between a pair of jobs i and j indicates that at least λ_{ij}^{FS} periods have to pass away after finishing job i before job j can be started. For example, concrete usually must dry for a number of days before building up walls or drilling holes into it.

Besides introducing minimum time lags, further types of precedence relationships are defined. A "start-to-start" relationship between job i and job j with a time lag λ_{ij}^{SS} is present when job j can not start before the processing of job i has been executed for at least λ_{ij}^{SS} time periods. In case of the time lag λ_{ij}^{SS} being smaller than duration d_i , the jobs i and j may be partially performed in parallel whereas in case of $\lambda_{ij}^{SS} \geq d_i$ this is not possible. However, in the first case, their execution may overlap by at most $d_i - \lambda_{ij}^{SS}$ periods. A "finish-to-finish" relationship with a time lag λ_{ij}^{FF} exists when job j can not be finished until at least λ_{ij}^{FF} periods have passed away after completing job i. Finally, a "start-to-finish" relationship with a lag λ_{ij}^{SF} specifies that there must be at least λ_{ij}^{SF} periods between the start of job i and finishing job j. When drawing a network diagram, the different types of precedence relationships are often represented as given in Figure 2.3 with the corresponding minimum time lags being denoted below the arcs. In particular, this representation is used within most project management software packages (cf. Section 2.5).

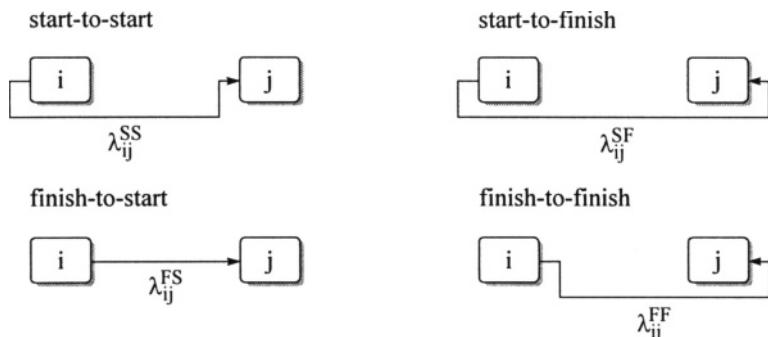


Figure 2.3. Graphical representation of precedence relations

A further extension of the precedence diagramming method consists of additionally considering *maximum time lags* between jobs. For example, in case of a "finish-to-start" relationship between two jobs i and j, the maximum time lag limits the number of periods which may pass between the termination of job i and the start of job j. Maximum time lags have first been introduced in the context of the metra-potential method (MPM) (cf. Roy (1964)). However, though there exist many possible modeling applications for maximum time lags, e.g., in make-to-order production, they have been discussed in the literature only rarely. A recent survey on applications as well as according methods is provided in Neumann and Schwindt (1997).

In general, minimum and maximum time lags of the different types can easily be transformed into each other. For example, a "finish-to-start" relationship with a minimum time lag of λ_{ij}^{FS} can be represented by a "start-to-start" relationship with a minimum time lag of $\lambda_{ij}^{SS} = d_i + \lambda_{ij}^{FS}$. Surveys on corresponding transformation rules are, e.g., given in Bartusch et al. (1988), Elmaghraby and Kamburowski (1992), and Domschke and Drexel (1998, p. 93). Furthermore, see Section 3.2.2.1, pp. 91. However, when considering such generalized precedence relationships the resulting project networks must no longer be acyclic. As a consequence, the simple critical path analysis as described in Section 2.2.1 can no longer be applied. Furthermore, anomalies in the computation of slack times may occur (cf. Section 2.2.2).

2.1.3 Activity-on-Arc Networks

Instead of choosing an activity-on-node representation, also a activity-on-arc network may be used for depicting the precedence relationships. As already stated earlier, when drawing the network, an arc instead of a node is used to represent a job. The head of the arc indicates the progress of the corresponding job. In order to introduce the precedence relationships between jobs, *events* are defined. An event corresponds to some time point at which a set of jobs is completed and another set starts. Thus, the beginning and the finishing points of a job (arc) are defined by two events referred to as *head* and *tail events*. The processing of jobs originating from an event can not be started before all jobs leading to this event have been completed.

The activity-on-arc network for the example of Table 2.1 is given in Figure 2.4. The numbers of the jobs as well as their durations are denoted on the arcs. The

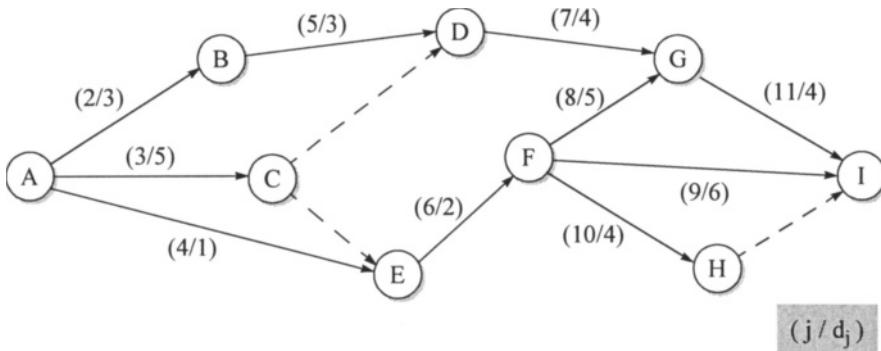


Figure 2.4. Activity-on-arc representation of the example project

nodes symbolize the events and are labeled alpha-numerically. The hatched arcs represent *dummy jobs*. Such jobs which do not consume time or resources are required to depict the precedence relationships correctly. Consider for example the jobs 3 and 5 which both precede job 7. However, job 3 additionally possesses the successor 6. Thus, if the corresponding arcs of job 3 and 5 terminate in the same node (event), job 5 will also become a predecessor of job 6. Therefore, a dummy job has to be introduced connecting the tail event of job 3 and the tail event of job 5. The same situation is present for the jobs 3 and 4 with job 3 having the additional successor job 7. The dummy job between the events H and I is introduced in order to avoid that two jobs which can be processed in parallel have identical head and tail events. Though this does not lead to logical errors within the network, it is considered to be bad drawing style.

Compared to activity-on-node networks, activity-on-arc ones show a number of disadvantages. First of all, their construction is rather difficult because dummy jobs have to be introduced. With an increasing number of such dummy jobs the distinctness of the network may considerably be reduced. This disadvantage becomes even more apparent taking into account that drawing an activity-on-arc network with a minimum number of dummy jobs represents an NP-hard optimization problem (cf. Section 3.2.3 as well as Dimsdale (1963), Krishnamoorthy and Deo (1979), Willis (1981), Syslo (1984), and Michael et al. (1993)). Furthermore, when varying networks for the same set of precedence relationships have been developed due to different use of dummy jobs, simple slack time calculations may not yield identical values for one and the same project (cf. Elmaghraby and Kamburowski (1990)).

Remark 2.2. From the presentational point of view, activity-on-node networks have the advantage that more evolved types of precedence relationships such as considered within precedence diagramming can easily be visualized. Furthermore, within these networks additional information on a job like, e.g., earliest and latest starting and finishing times, can easily be included within a node which may be confusing when given next to an arc. Due to these reasons, most project management software packages prefer the activity-on-node representation. For the same reasons, the following expositions are also restricted to this type of project network. Nevertheless, activity-on-arc networks may still represent the more natural approach when emphasis is on events rather than on jobs as this is, e.g., the case within milestone planning and project crashing.

2.2 Scheduling

As already stated in Section 1.5.2, scheduling is concerned with establishing dates during which the jobs required for completing the project will be executed. For this purpose, a critical path analysis as well as slack time computations are performed on the basis of the project network obtained during the structuring process. Based on these results, a schedule may be set forth by the project manager. Most commonly, this schedule is represented in form of a Gantt chart in order to efficiently communicate it among the participants of the project.

2.2.1 Critical Path Analysis

The primary goal of the critical path analysis is to determine the smallest possible project completion time which can be realized without violating the precedence constraints imposed by the project network. Furthermore, earliest and latest starting and finishing times are computed. For each job, they define a time window during which it has to be processed when the project is to be completed as soon as possible. In the following, the application of the critical path analysis to acyclic precedence networks with "finish-to-start" relationships omitting minimum and maximum time lags is described. To ease the presentation, the jobs are assumed to be numbered following a *topological ordering*, i.e., $i < j$, if job i is predecessor of job j . For complex project networks, it may not be obvious how to label the nodes in the desired manner. An according algorithm is given in Shtub et al. (1994, p. 332).

Within a *forward pass*, earliest starting and finishing times ES_j and EF_j for all jobs $j = 1, \dots, n$ are determined at which the jobs may be scheduled the earliest observing the precedence relationships. Furthermore, the smallest possible project completion time CT which equals the earliest finishing time EF_n of the terminal dummy job n is computed. Starting with $ES_1 = EF_1 = 0$, the values of the jobs $j = 2, \dots, n$ are obtained as follows:

$$ES_j = \max\{EF_i \mid i \in P_j\} \quad \text{for } j = 2, \dots, n \quad (2.1)$$

$$EF_j = ES_j + d_j \quad \text{for } j = 2, \dots, n \quad (2.2)$$

Subsequently, latest finishing and starting times LF_j and LS_j are computed for all jobs $j = n, \dots, 1$ by a *backward pass*. They denote when jobs can be scheduled the latest in order to realize the earliest project completion time CT without violating the precedence constraints. For each job j , the interval $[ES_j, LF_j]$ defines a *time window* during which j has to be processed if the project should be completed after CT periods. Beginning with $LF_n = LS_n = CT$, the following computations are performed:

$$LF_j = \min\{LS_h \mid h \in F_j\} \quad \text{for } j = n-1, \dots, 1 \quad (2.3)$$

$$LS_j = LF_j - d_j \quad \text{for } j = n-1, \dots, 1 \quad (2.4)$$

Remark 2.3. Note that when a given project deadline exists which exceeds the smallest possible completion time CT , the latest finishing and starting time of the terminal dummy job are set to this value for computing according time windows.

Table 2.2. Earliest and latest starting and finishing times for example project

j	1	2	3	4	5	6	7	8	9	10	11	12
ES_j	0	0	0	0	3	5	6	7	7	7	12	16
EF_j	0	3	5	1	6	7	10	12	13	11	16	16
LS_j	0	2	0	4	5	5	8	7	10	12	12	16
LF_j	0	5	5	5	8	7	12	12	16	16	16	16

The results obtained by a forward and a backward pass for our example of Figure 2.2 are given in Table 2.2. The project can be completed after 16 periods the earliest. In general, the smallest possible completion time of a project is determined by a longest path through the project network. This path connecting

the dummy start and end node is called *critical path*. Depending on the precedence relationships to be considered, eventually several critical paths may exist which can partially overlap. In our example, there is only one critical path which is defined by the jobs $CP = \langle 1, 3, 6, 8, 11, 12 \rangle$.

Besides computing the smallest possible project completion time by performing a forward pass, a linear optimization problem may be formulated. With ST_j denoting the starting time of a job j , the problem may be stated as follows (cf., e.g., Carruthers and Battersby (1966), Elmaghraby and Kamburowski (1992), and Domschke and Drexl (1998, pp. 101)):

$$\text{Minimize } CT(ST) = ST_n \quad (2.5)$$

subject to

$$ST_i + d_i \leq ST_j \quad \text{for } j = 2, \dots, n \text{ and all } i \in P_j \quad (2.6)$$

$$ST_1 = 0 \quad (2.7)$$

The objective function (2.5) minimizes the starting time ST_n of the dummy end job and, hence, the project completion time. The constraints (2.6) ensure that the precedence relationships are observed. Finally, constraint (2.7) guarantees that the project is started at the beginning of the planning horizon. Note that the values ST_j obtained for the jobs $j = 2, \dots, n - 1$ may not correspond to the earliest starting times ES_j calculated by the forward pass. In order to obtain identical values, a penalty term $\epsilon \cdot \sum_{j=2}^n ST_j$ has to be added to the objective function with $\epsilon > 0$ being a constant.

Though in general the forward and the backward pass are used for the critical path analysis due to being more efficient, the formulation of an optimization problem may be useful when additional constraints have to be observed. For example, time lags as considered by precedence diagramming can easily be incorporated and it does not matter whether the precedence network is acyclic or not. For specialized procedures which are able to deal with acyclic project networks see Elmaghraby and Kamburowski (1992), Neumann and Schwindt (1997), as well as Domschke and Drexl (1998, section 5.2.2).

Several other extensions of the critical path method have been considered in the literature. An approach which addresses problems that arise from considering a calendar with non-working days is described in Zhan (1992). The effect of vari-

able job durations is examined in Leachman and Kim (1993). Time window and time schedule constraints are introduced by Chen et al. (1997). Time window constraints are present when a job can only be executed in a certain time interval. Time schedule constraints assume that the execution of jobs may only start at prespecified time points.

2.2.2 Slack Time Computations

As described above, the smallest possible completion time of a project corresponds to the length of a *critical path* through the corresponding network. The execution of jobs defining such a path may not be prolonged or postponed without delaying the termination of the project beyond the smallest completion time possible and, hence, these jobs are also termed *critical*. However, those jobs which are not located on a critical path possess a *slack time* or *float* and can be prolonged or postponed within certain limits. Calculating the slack time of jobs provides valuable information both for project planning and control. Within project planning, rearranging non-critical jobs within their slack times may help to develop a schedule which meets budget and resource constraints. Furthermore, in a multiproject environment, slack available in one project may be used to temporarily free resources which are required by other projects. Within project control, particular emphasis can be made on critical and near-critical jobs in order to ensure that the project is completed on schedule.

Most commonly, four different types of slack time are distinguished in the literature which have been introduced by Battersby (1967) and Thomas (1969):

- *Total Slack Time*. It is defined as the number of periods by which the execution of a job may be postponed from its earliest starting time or by which its duration may be prolonged without increasing the smallest possible project completion time. The total slack time TSL_j of a job j is computed as the difference of the latest and earliest starting time of the job:

$$TSL_j = LS_j - ES_j \quad (2.8)$$

By computing the total slack time, the critical jobs for which $TSL_j = 0$ is true can easily be determined.

- *Free Slack Time*. The free slack time FSL_j of a job j corresponds to the number of periods by which the execution of job j can be delayed without

postponing the earliest starting time of its followers $i \in F_j$. It is computed as follows:

$$FSL_j = \min\{ES_i \mid i \in F_j\} - EF_j \quad (2.9)$$

- *Safety Slack Time.* The safety slack time SSL_j of a job j represents the number of periods by which the duration of job j can be prolonged when all its predecessors $h \in P_j$ are started as late as possible without increasing the smallest possible project completion time. It is calculated by the following formula:

$$SSL_j = LS_j - \max\{LF_h \mid h \in P_j\} \quad (2.10)$$

- *Independent Slack Time.* The independent slack time ISL_j of a job j equals the number of periods by which the duration of job j can be prolonged irrespective of the completion times of its predecessors P_j and the starting times of its followers F_j . It is obtained as follows:

$$ISL_j = \max\{0, \min\{ES_i \mid i \in F_j\} - \max\{LF_h \mid h \in P_j\} - d_j\} \quad (2.11)$$

The slack time values which are yielded by computing the different types of slack time to the example project of Figure 2.2 are given in Table 2.3. For all jobs defining the critical path $CP = \langle 1, 3, 6, 8, 11, 12 \rangle$ the total slack time is $TSL_j = 0$. In the table, these jobs are indicated by the shaded cells. Since the relationships $TSL_j \geq FSL_j \geq ISL_j$ and $TSL_j \geq SSL_j \geq ISL_j$ hold, all other slack time values of these jobs are 0, too. Job 4 shows identical values for all types of slack time because all its predecessors and all its successors are located on the critical path. The same is true for the jobs 9 and 10, respectively.

Table 2.3. Slack times for the example project

j	1	2	3	4	5	6	7	8	9	10	11	12
TSL_j	0	2	0	4	2	0	2	0	3	5	0	0
FSL_j	0	0	0	4	0	0	2	0	3	5	0	0
SSL_j	0	2	0	4	0	0	0	0	3	5	0	0
ISL_j	0	0	0	4	0	0	0	0	3	5	0	0

Though these slack time definitions have been widely accepted, they show some disadvantages. In generalized project networks considering minimum and maximum time lags, the critical jobs can no longer be determined by simply

choosing the jobs which do not possess a total slack time. Furthermore, different types of critical jobs have to be distinguished and slack time computations have to be adapted (cf., e.g., Wiest (1981), Ziegler (1985), and Elmaghraby and Kamburowski (1992)). Furthermore, if an activity-on-arc representation is chosen, the definitions of the different slack time types have to be modified and can lead to different values depending on how dummy jobs are introduced (cf. Elmaghraby and Kamburowski (1990)).

In case of resource constraints being present, the determination of valid slack times for jobs becomes much more difficult. This is due to the fact, that in this case also relationships between jobs which are not connected by a precedence relationship have to be considered. However, this is only possible for a completely or at least partially defined schedule. Initial work on this field has been performed by Wiest (1964). Further approaches have been presented by Willis (1982), Woodworth and Shanahan (1988), Bowers (1995), and Raz and Marshall (1996).

Within the context of slack time computations, another problem may arise. In case of the execution of a non-critical job being postponed or its duration being prolonged, its total slack time may be used to compensate for the delays. However, this may result in some of its followers becoming critical (cf., e.g., Matthes (1980)). In our example, this is, e.g., true for the jobs 5 and 7 if the duration of job 2 is extended by 2 periods. In case that also the durations of one of these two jobs is prolonged, the project completion time increases. In order to avoid such situations, the total slack time available for a subset of jobs has to be distributed among them according to some criteria. According procedures have, e.g., been proposed by Williams (1978) and Popescu and Pasiphol (1995). This problem becomes even more important, if a project is performed on the request of a customer. Then, often the customers try to delay non-critical jobs in order to also postpone according progress payments (cf., e.g., Love (1983)). However, if these jobs are subject to delays which lead to an increase of the project completion time, the question arises who is responsible for the delay. Therefore, the "ownership" of slack time should already be defined within the project contract (cf., e.g., Householder and Rutland (1990) and De La Garza and Vorster (1991)). Recognizing this fact, critical path analysis is more and more used for identifying causes for delaying a project in case of contractual issues (cf. Tavakoli and Riachi (1990) and Kallo (1996)).

2.2.3 Gantt Charts

Having performed the critical path analysis, the data is available which is required for establishing a project schedule. A *schedule* assigns starting and finishing times to all jobs. Most often, this is done following an *early starting time approach*, i.e., each job is scheduled to start at the earliest starting time which has been calculated by the forward pass. This follows the rationale, that if the execution of a job is unpredictably delayed a maximal amount slack time is available. However, due to resource or budget constraints also other starting times may have to be established (cf. Section 2.3). Since the so-defined schedule may be subject to changes during the execution of the project, it usually is referred to as *baseline schedule* to distinguish it from the current one.

Subsequently, the schedule defined is commonly visualized as a *Gantt chart* which is one of the most widely used management tools for project scheduling and control. In a modified form of a bar chart, the jobs to be executed are enumerated on the vertical axis whereas the time intervals during which they are processed are depicted on the horizontal axis. The Gantt chart in Figure 2.5 shows the schedule which is obtained for our example by an early starting time approach (cf. Table 2.2). For example, the processing of job 8 with $ES_8 = 7$ is started after 7 periods (at the beginning of period 8) and finishes after 12 periods. Those jobs which are critical are emphasized by shaded bars.

In the literature as well as in practice, a large number of extensions of this basic Gantt chart type are considered in order to convey additional information. For example, the total slack of the jobs is represented by dashed bars attached at the end of the bars symbolizing the execution of a job. In particular, in project management software packages, the precedence relationships are integrated into the Gantt chart by linking the appropriate bars. When different types of precedence relationships are depicted, they are usually drawn as given in Figure 2.3. Furthermore, special symbols exist in order to introduce events like milestones or reviews into the Gantt chart or to visualize the status of a job, e.g., whether it is completed or delayed (cf., e.g., Lee-Kwang and Favrel (1988)). However, the problem with these extensions is that they take away the distinctness of the basic approach. Therefore, Gantt charts and network diagrams are often used simultaneously. For example, most commercial project management software packages allow to switch between both representations.

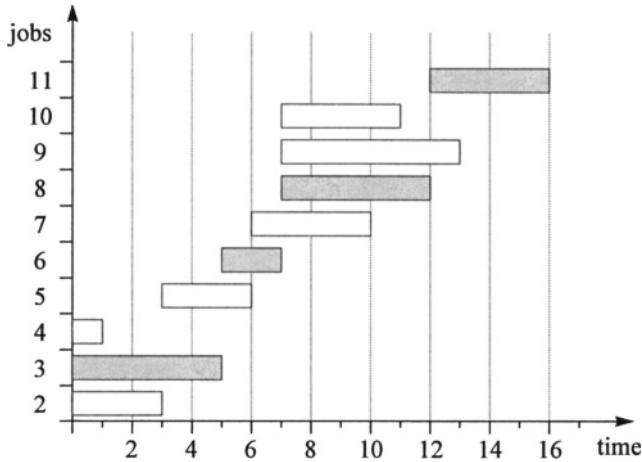


Figure 2.5. Gantt chart for example project

2.3 Resource Allocation

In the descriptions given so far, the only constraints for the construction of a feasible schedule are assumed to be imposed by the precedence relationships. However, as indicated in Section 1.5.3, a schedule is subject to a number of additional constraints, e.g., due to the defined budget or the restricted availability of resources. In particular, the latter type of constraints are important within the planning of the project's execution because the availability of resources may usually not be altered considerably within the restricted planning horizon of several weeks or months. In this section, we describe how to integrate such constraints into the process of developing a schedule.

2.3.1 Resource Loading

In order to examine the effects of the resource constraints on a schedule, the utilization of the different resource types involved in the execution of the project have to be determined for each period of the planning horizon. If in some period the utilization of a resource type exceeds the respective availability, a *resource conflict* occurs which has to be resolved (cf. Sections 2.3.2 and 2.3.3). Most commonly, only *renewable* resource types which are available in each period of the planning horizon by a prespecified amount are considered for detect-

ing possible resource conflicts. To ease the presentation, we introduce the following notation (cf. Section 3.2.1).

The processing of a project involves m renewable resource types defined by the index set $R = \{1, \dots, m\}$. In the most simple form of resource allocation, a type r is assumed to be constantly available by an amount of a_r capacity units ($r = 1, \dots, m$) in each period of the planning horizon. Instead of considering only constant resource availabilities, also the case of dynamic ones changing from period to period may be considered (cf. Section 3.2.2). Predictable fluctuations in the availability of a resource type may, e.g., be due to vacancies of a project team member or to maintenance of equipment. The execution of a non-dummy job j ($j = 2, \dots, n - 1$) requires u_{jr} units of resource type r in each period in which it is processed.

Having provided these data as well as a schedule, the *resource loading* of each resource type, representing its utilization over time, may be computed as follows. For each period of the planning horizon, the jobs in progress are determined. Subsequently, for each resource type, the total demand of these jobs is calculated by summing up their individual per period resource usages.

Table 2.4. Resource usages of example project

j	1	2	3	4	5	6	7	8	9	10	11	12
u_{j1}	0	2	3	3	1	1	2	3	1	1	1	0

Table 2.4 shows the resource usages which have been estimated for the jobs of the example problem in Figure 2.2. Only a single resource type $r = 1$ with a constant per period availability of $a_1 = 4$ is considered. Assuming that the schedule depicted in Figure 2.5 is to be realized, the resource loading of the periods $t = 1, \dots, 16$ can be calculated as follows. In the first period, the jobs 2, 3, and 4 with resource usages of $u_{21} = 2$, $u_{31} = 3$, and $u_{41} = 3$ are processed. Hence, their total demand equals 8 capacity units which exceeds the availability $a_1 = 4$ by 4 units and causes a resource conflict. In the second period, job 4 has been terminated but no other job has been started yet due to the precedence constraints. Thus, the resource loading reduces to 5 capacity units. The resource utilization of the subsequent periods can be computed accordingly.

After calculating the loading of a resource type for all periods of the planning horizon, its *loading profile* may be drawn depicting the utilization of a resource type as a function of time. Within project management software packages, the

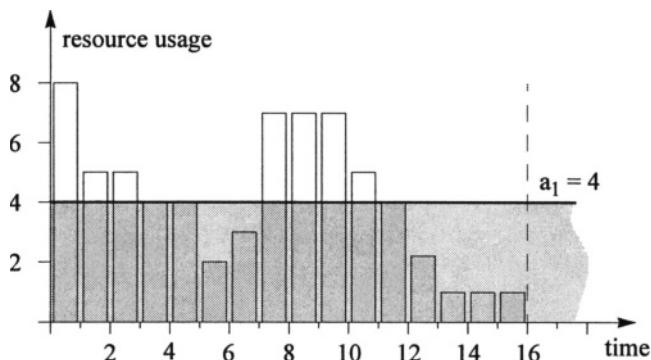


Figure 2.6. Resource loading profile

loading profile is commonly visualized in form of a histogram as given in Figure 2.6. On the vertical axis, the resource usage is represented whereas the periods of the planning horizon are denoted on the horizontal one. Furthermore, the resource availability is depicted such that resource conflicts can be detected. In general, this type of representation has the disadvantage that the jobs which cause a resource conflict can not be recognized.

Alternatively, *capacity-oriented* Gantt charts may be developed for each resource type. In contrast to the presentation described above, the non-dummy jobs $j = 2, \dots, n - 1$ are depicted by rectangles with horizontal extents d_j and vertical extents u_{jr} . According examples are given in the Figures 2.7 and 2.8. However, the development of such charts is more complex than the preparation of histograms.

2.3.2 Resource-Constrained Scheduling

Determining the resource loading profile for a given schedule may lead to the detection of resource conflicts. These conflicts have to be resolved in order to obtain a realizable schedule. The most simple form of resolution consists of allocating additional resources in the periods affected which might not always be possible due to the resulting additional cost exceeding the approved budget and reducing the project's profit. Furthermore, the availability of resource types like equipment and machines can eventually not be increased within a short time because of technological restrictions or lack of skilled workers.

Facing such a situation, the objective of *resource-constrained scheduling* usually consists in developing a schedule such that the project is completed as early as possible considering both the precedence relationships and the restricted availabilities of resources. For example, it might be possible to modify the starting times of non-critical jobs within their slack times such that the resource conflicts are resolved. If this fails, it has to be decided which of the jobs involved in the remaining resource conflicts have to be postponed beyond their latest starting time (cf. Section 1.5.3).

In general, solving the resulting *resource-constrained project scheduling problem* (RCPSP) represents a difficult task, in particular when several resource types have to be considered. Therefore, this problem has extensively been studied in the literature during the last decades. A large number of procedures have been developed which are able to solve the problem either optimally or heuristically. In this book such procedures are reviewed as well as new ones improving the existing ones are described such that a extensive discussion is omitted at this point (cf. Chapters 4 to 6 for detailed presentations).

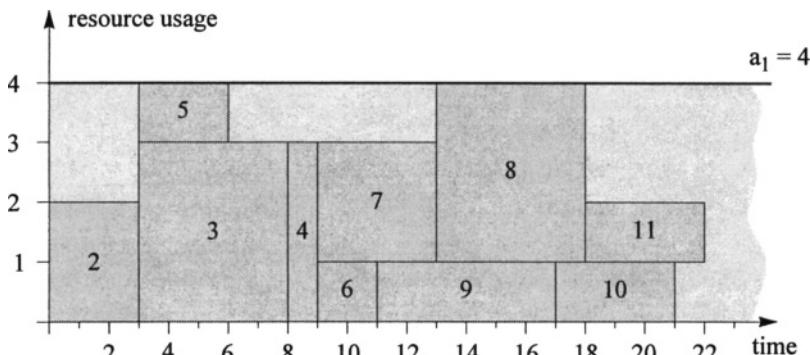


Figure 2.7. Optimal resource-constrained schedule

Figure 2.7 shows an optimal solution of the resource-constrained project scheduling problem defined by the data given in Figure 2.2 and Table 2.4 in form of a capacity-oriented Gantt chart. Assuming a single resource type $m = 1$ with a per period availability $a_1 = 4$, job 2 is performed first. None of the other jobs can be processed simultaneously, because the jobs 3 and 4 require too much resources and the remaining jobs are successors of the jobs 2, 3, and 4. After finishing job 2, its successor 5 and job 3 can be performed concurrently for 3 periods. Then job 5 is finished and job 3 is continued for two further periods solely.

Subsequently, job 4 is also executed exclusively. Afterwards, the job pairs (6,7), (7,9), (8,9), (8,10), and (10,11) which are not connected by precedence relationships, respectively, can be performed in parallel. Since the dummy end job can immediately start (and terminate) after finishing job 11, the project is completed after 22 periods.

2.3.3 Time-Constrained Scheduling

A problem which is closely related to resource-constrained scheduling arises when a strict deadline on the project completion time CT is given, e.g., due to a contract defining high penalty payments. In this case, the only possibility of resolving eventual resource conflicts consist of rearranging the jobs within their slack times. If this is not sufficient to resolve all resource conflicts, the availabilities of the affected resource types have to be temporarily increased in the corresponding periods. This can, e.g., be done by approving overtime work or lending equipment. However, as described above, this results in additional cost. In order to keep them as small as possible, *time-constrained scheduling* is concerned with the rearrangement of non-critical jobs within their slack times such that the total additional cost are minimized and the project termination is not delayed (cf. Section 3.4.1).

For this purpose, the per period cost o_r in monetary units which arise for providing an additional unit of each resource type $r = 1, \dots, m$ have to be estimated. Then, the total additional cost for a given schedule can be calculated as follows. For each period t , the jobs currently being processed are determined and their total demand of each resource type is computed. If for a resource type r the total resource demand exceeds the availability in the period t by w_{rt} capacity units, the cost of extending the availability in this period equals $o_r \cdot w_{rt}$. Finally, the additional cost for completing the project within the time limit of CT units can be determined by $\sum_{t=1}^{CT} \sum_{r=1}^m o_r \cdot w_{rt}$.

Figure 2.8 shows an optimal time-constrained schedule for the example project defined by the data of Table 2.2 and Table 2.4 assuming that it has to be finished after $CT = 17$ periods the latest. Since only a single resource type with an availability of $a_1 = 4$ is considered, the solution does not depend on the actual value of o_1 . The earliest starting and finishing times of the jobs equal those given in Table 2.4. By the way of contrast, the latest starting and finishing times

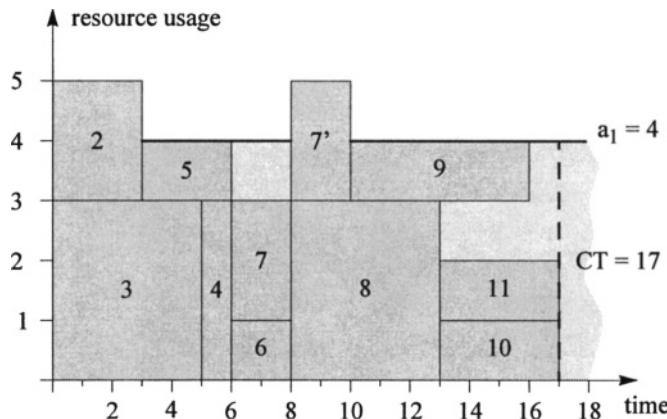


Figure 2.8. Optimal time-constrained schedule

are increased by one period due to the backward pass being started with $LF_n = LS_n = 17$ instead of the critical path length 16.

The jobs 2 and 3 are immediately started after the beginning of the project whereas job 4 is delayed within its slack time in order to avoid additional cost. After finishing job 2, job 5 is put into progress terminating together with job 4. Subsequently, the jobs 6 and 7 are immediately scheduled. Also, the execution of job 8 is begun as soon as possible, i.e., after completing job 6, in order not to increase the project completion time. However, the jobs 9 and 10 can be postponed preventing from further additional cost. Finally, job 11 has to be executed. For the resulting schedule, a total of 5 additional capacity units has to be provided.

2.4 Control

Project control represents the major management task during the execution of the project. Basically, there are three different dimensions of project control. First of all, *schedule control* is concerned with comparing the baseline schedule to the actual performance of the project. Second, *cost control* has to detect and avoid potential cost overruns. For this purpose, usually an earned value analysis is executed (cf. Section 2.4.2). Finally, *performance control* has to ensure that the outcome of the project corresponds to the specifications made in the defini-

tion phase which can, e.g., be achieved by using quality management tools such as reviews and audits (cf. Section 1.6.3).

2.4.1 Schedule Control

For performing schedule control, data on the progress of the project has to be monitored either periodically (e.g., every week or month) or continuously (e.g., each time a job has been completed or a milestone has been achieved). Note that the *control policy* which is chosen for determining according checkpoints and for defining the intensity of monitoring has a considerable influence on the potential success of project control. For example, within a end-loaded policy, monitoring is accomplished less intensive in the early phases of the project and is successively increased towards the completion of the project. Within a front-loaded policy, this is just the other way round. A comprehensive discussion of different monitoring policies is, e.g., contained in Partovi and Burton (1993). By a simulation experiment they show that avoiding delays of the project completion time is most likely when an end-loaded policy is applied.

Irrespective of the monitoring policy chosen, data on the progress of the project is usually collected by measuring the percent degree by which each job is completed at a prespecified checkpoint t . Having these data available, they can be compared to the values which have been set forth for this checkpoint in the baseline schedule. Most commonly, this is done graphically by drawing *progress Gantt charts*. One of the most popular forms of a progress Gantt chart is given in Figure 2.9 which extends the one of Figure 2.5. The degree by which a job is completed is depicted by shading the according bar appropriately. Considering, e.g., job 8, only 20% of its work content has been accomplished until the current checkpoint $t = 10$. However, as the Gantt chart shows, already 60% of the work content should have been performed according to the baseline schedule. Since job 8 is critical, a delay of the project completion time will occur, if its execution can not be expedited. By the way of contrast, job 10 is processed on schedule. Finally, Job 9 is even executed faster than initially planned.

Besides this type of progress Gantt chart, which is commonly used within project management software packages, a large variety of other types exist (cf., e.g., Burke (1992, section 11.6)). In case that an adapted schedule has been developed in order to put the project back on course, often the baseline schedule as well as the current one are depicted in a single Gantt chart. For this purpose,

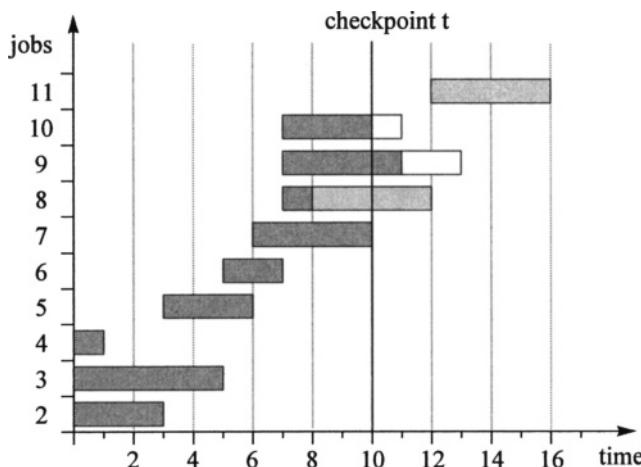


Figure 2.9. Progress Gantt chart for example project

two bars are drawn for each job. The first one represents the executions dates defined in the baseline schedule, whereas the second one visualizes the currently planned dates.

With an increasing number of jobs, drawing progress Gantt charts may constitute a considerable effort, if it is not supported by appropriate computer software. Furthermore, the charts may lack of distinctness if only the progress of the project as a whole has to be represented, e.g., within reports for the senior management. Therefore, often *progress plots* are additionally developed (cf., e.g., Schmidt (1988), Tavakoli (1990)) which concentrate on the jobs of a critical path. In a two-dimensional coordinate system, the vertical axis denotes by which degree the jobs on this path have been completed by giving the accomplished work content of the path in percent. The horizontal axis represents the planning horizon during which the project is executed. Then, the planned progress of the project can be visualized by a line connecting points depicting the relative work content to be completed up to a certain period. For example, job 8 which lies on the critical path of the example project in Figure 2.2 may be scheduled to finish after 12 periods. Since the length of the critical path equals 16 periods, 75% of the critical path's work content have been accomplished after completing job 8. To enhance the readability of the chart, the completion of the jobs on the critical path is usually visualized by according symbols.

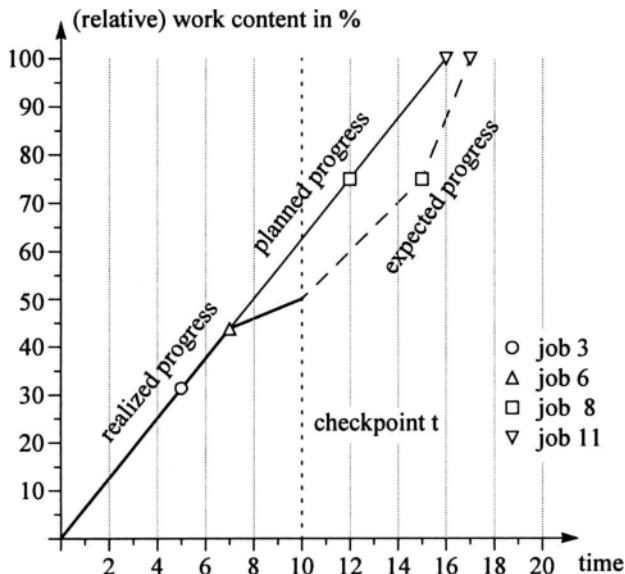


Figure 2.10. Progress plot

Having depicted the planned performance, which is a straight line, the realized progress of the project is represented accordingly. At each checkpoint t , the degree of the completion of the critical jobs is determined. Following the above example, job 8 is only completed by 20% at the checkpoint $t = 10$, a degree of completion which should have already been achieved after 8 periods. Hence, only 50% of the project's work content have been completed instead of 60% as initially planned. As a consequence, the bold line depicting the realized progress falls below the one showing the planned progress. This indicates that the project will not finish on time. In the example, we assume that job 8 is predictably further delayed and does not finish before period 15. Thus, at the end of period 15 only 75% of the project's work content have been accomplished instead of the scheduled 95%. If it is possible to expedite the delayed job or one of its successors on the critical path, the realized progress line will come closer to the planned one again. For example, due to detecting the possible delay of the project already at the checkpoint $t = 10$, it might be possible to allocate additional resources to job 11 such that its duration is reduced to 2 periods and the project terminates after 17 periods.

A major disadvantage of this representation is that it solely concentrates on the jobs of a critical path. If near critical jobs are delayed or postponed another path

through the project network may become critical instead of the current one. Furthermore, several critical paths may exist. Finally, if a project completion time exceeding the critical path length is planned, e.g., due to restricted resource availability, identifying the jobs to be considered for measuring the progress is difficult.

In the main, the two control instruments presented so far are suited for detecting delays or postponements of jobs which have already occurred. However, this may be too late to put the project back to course such that it finishes on time. Therefore, possible delays and postponements which endanger the on schedule completion of the project have to be anticipated. For this purpose, often a *trend analysis* is performed (cf. Burghardt (1997, section 4.1.4)). The basic idea consists of periodically estimating the time of achieving advised milestones or finishing important jobs which either are in progress or are still to be started. If, e.g., the expected finishing time of a job is increased once, this may usually be compensated during the remainder of the project. However, if it is postponed repeatedly, i.e., from estimation to estimation, this may really lead to an increase of the project completion time.

Though a trend analysis may refer to any job with a scheduled execution date, usually only those jobs are considered the on schedule execution of which is important for successfully completing the project on time. These may, e.g., be jobs located on a critical path. In order to be able to extrapolate a trend from the estimates made at different checkpoints, they are depicted graphically as given in Figure 2.11. The vertical axis represents the planning horizon of the project, whereas the horizontal one denotes the control horizon. Each time an estimate is made for a job, a new mark is inserted into the diagram at the coordinate representing the period of the estimation as well as the expected completion time of the job. By connecting these marks for each job, a trend curve for the expected completion time of each considered job may be derived. If the curve of a job develops horizontally, the job will probably be finished on schedule. However if the curve rises, the completion of the job will presumably be delayed. A falling curve indicates that a job will be terminated earlier than expected.

Figure 2.11 shows a trend analysis for the example project of Figure 2.2, assuming that each job is started at its earliest starting time determined by the critical path analysis. In the example, the expected finishing times of the critical jobs are estimated every two periods. At the beginning of the project, the estimates correspond to the scheduled finishing times. For the next two estimations

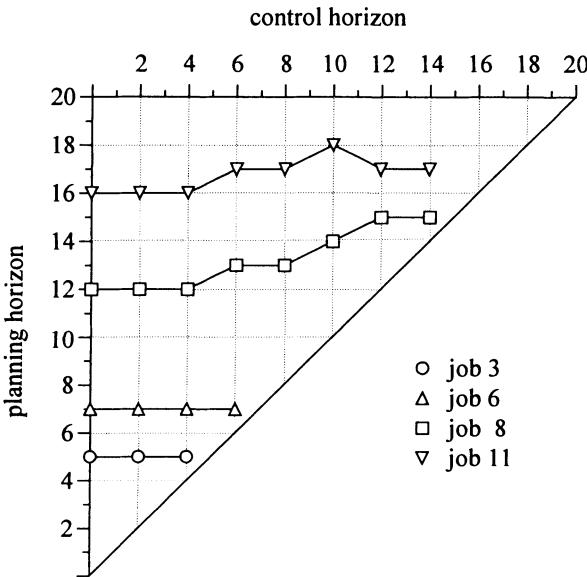


Figure 2.11. Trend analysis

at $t = 2$ and $t = 4$, this is true, too. In period 6, the finishing time of job 8 is expected to be delayed by one period. As a consequence, also the execution of job 11 has to be postponed by one period. Further delays are detected in the periods 8 and 10, respectively. However, as assumed within the example for the progress plot, additional resources are provided for the execution of job 11 such that its execution can be expedited and it is terminated already after period 17.

2.4.2 Cost Control

As described earlier, cost control constitutes the second dimension of project control which has to be addressed during the execution of the project. Within cost control, monitoring the required data is a major problem. This is due to conventional cost monitoring systems providing the data periodically, e.g., at the end of a month. However, for project cost control, these costs have to be divided upon the jobs executed in this period. Therefore, usually special cost monitoring systems have to be established (cf. Section 1.6.1).

Most commonly, simple project cost control restricts to determining the actual accumulated cost for the work performed until the current period. This value is compared to the one budgeted during the planning phase. Restricting to such simple types of cost control may result in wrong assessments on the actual cost

performance of the project. Therefore, an *earned value analysis* may additionally be executed. The basic idea of this concept is to integrate cost and schedule control by appraising the work performed using monetary values. For a comprehensive introduction into this concept we refer to, e.g., Burke (1992, chapter 12), Lambert (1993), and Lewis (1995).

Table 2.5. Cost for performing jobs of the example project

j	1	2	3	4	5	6	7	8	9	10	11	12
tc _j	0	90	100	20	30	40	60	150	90	80	60	0

The problem of simple cost control as well as the concept of the earned value analysis can best be explained by extending the example of Figure 2.2. For this purpose, the expected total cost tc_j (in monetary units) for performing each job j are introduced (cf. Table 2.5). Based on a given schedule, the *budgeted cost of work scheduled* $BCWS_t$ can be calculated. It corresponds to the accumulated value (in monetary units) of the work scheduled to be accomplished until the end of a period t . For the computation of $BCWS_t$, all jobs already finished are considered with their total cost tc_j . The cost for jobs still in progress are determined by multiplying their total cost tc_j by the degree of their completion in percent.

Table 2.6. Budgeted cost of work scheduled for the example project

t	1	2	3	4	5	6	7	8
BCWS _t	70	120	170	200	230	260	295	375

t	9	10	11	12	13	14	15	16
BCWS _t	455	535	600	645	675	690	705	720

Considering the schedule depicted in Figure 2.5, p. 50, the values $BCWS_t$ obtained at the end of the periods $t = 1, \dots, 16$ are given in Table 2.6. For example, at the end of period 10, the jobs 2 to 7 will be completed resulting in accumulated cost of 340. By the way of contrast, the jobs 8, 9, and 10 will have been processed only partially, i.e., for 3 periods, respectively. Since job 8 has a duration of $d_8 = 5$, it is assumed to be completed by $3/5 \cdot 100\% = 60\%$. The corresponding induced cost are $150 \cdot 60\% = 90$. For the jobs 9 and 10, for which 50% and 75% of their work content will have been executed, the induced cost are 45 and 60, respectively. Hence, $BCWS_{10} = 340 + 90 + 45 + 60 = 535$.

During the execution of the project, the *actual cost of work performed* ($ACWP_t$) is determined by the monitoring process. It is defined as the accumulated cost incurred in performing the work accomplished until the end of the current period t . It may differ from the budgeted cost of work for several reasons. For example, the expected cost of completed jobs may have been estimated too high or too low. Furthermore, jobs still being processed may not be finished to the percent degree initially planned such that less cost have incurred so far. Vice versa, the execution of jobs may proceed faster than scheduled thus inducing corresponding cost earlier.

Consider, e.g., that at the end of period 10 the project has been processed as shown in Figure 2.9, p. 57. Furthermore, assume that the actual total cost arising for the execution of job 3 have been estimated too low, i.e., a total cost of 140 instead of 100 have incurred. As a consequence, the cost for the completed jobs at $t = 10$ rises from 340 to 380. Additionally, the costs induced by the jobs 8 and 9 which have only partially been processed change in comparison to the budgeted ones. Only 20% of job 8 have been executed instead of the scheduled 60% and no additional cost have yet occurred due to this delay. Hence, job 8 causes cost of only 30 instead of 90. Vice versa, job 9 is performed faster than initially expected. Due to this reason, the associated cost rise from 45 to 60. That is, we obtain $ACWP_{10} = 380 + 30 + 60 + 60 = 530$. Subsequently comparing this value to $BCWS_{10} = 535$ results in the wrong assessment that the project is performed on budget. However, the saving achieved is only due to the delay of job 8.

In order to avoid such misassessments, the *budgeted cost of work performed* ($BCWP_t$), also known as the *earned value*, is calculated. It corresponds to the budgeted monetary value of the work actually accomplished until the current period t . Hence, when computing $BCWP_t$, the total cost of the completed jobs are given by the sum of their budgeted cost tc_j instead of their actual ones. In our example, this results in ignoring the unexpected additional cost for job 3. When determining the cost of the jobs in progress, their degree of completion is considered. Since job 8 and job 9 are processed faster and slower than planned, respectively, the cost of the active jobs 8, 9, and 10 totals 150 as described above. Hence, we yield $BCWP_{10} = 340 + 150 = 490$.

Having computed the three measures $BCWS_t$, $ACWP_t$, and $BCWP_t$ at the end of a current period t , they can be compared to each other in order to detect deviations from the baseline schedule and the baseline budget.

In a period t , deviations from the baseline budget may be recognized by contrasting the values of $BCWP_t$ and $ACWP_t$. For this purpose, the *cost variance* CV is defined as the difference between $BCWP_t$ and $ACWP_t$. A positive variance shows that the cost is lower than the original estimate, whereas a negative one indicates that a cost overrun will occur. In the example for $t = 10$, the cost variance is $CV = 490 - 530 = -40$ caused by the additional cost for job 3. However, the cost variance represents an absolute measure such that it is difficult to assess the degree of the deviation between the scheduled performance and the actual progress. Therefore, to get a better impression, often a so called *cost index* is computed by $CI = ACWP_t/BCWP_t$ (cf., e.g., Shtub et al. (1994, pp. 471)). For our example, we obtain $CI = 530/490 = 1.08$. That is, the current cost overrun is approximately 8%.

By computing the difference between the values of $BCWP_t$ and $BCWS_t$, the *schedule variance* SV is determined. A positive schedule variance shows (in monetary units) that the work content performed exceeds the one scheduled until the current period t . By the way of contrast, a negative value implies that the project presumably will run late. This is, e.g., true for our example with $t = 10$ where the schedule variance equals $SV = 490 - 535 = -45$. Again, this measure is not very expressive due to being absolute. Therefore, the *schedule index* SI may be calculated by $SI = BCWP_t/BCWS_t$. In our example, we obtain $SI = 0.92$. That is, only 92% of the work content (in monetary units) initially to be performed until the end of period 10 has actually been realized.

2.5 Project Management Software

Project planning and control involve fulfilling a number of challenging managerial tasks. Therefore, soon after the introduction of computers in industrial and public organizations, first attempts were made to develop software packages supporting the accomplishment of these tasks. Early software packages mainly restricted on the scheduling and budgeting of projects. Both, the input of the data as well as its processing were usually batch-oriented resulting in a number of prespecified reports. Furthermore, those systems were commonly proprietary, i.e., developed by the organizations for their internal use only. The situation changed with personal computers broadly becoming available at the beginning of the eighties. Since then, a large number of integrated software packages have been developed covering a large variety of managerial tasks to

be addressed in the context of project management. Currently, more than 400 products can be purchased on the market differing considerably concerning their prices and capabilities. Due to this market being very dynamic, we refuse giving descriptions of particular products. Instead we refer to computer magazines or project management journals where such descriptions as well as competitive comparisons can be found regularly (cf., e.g., IIE Solutions or PM Network for yearly surveys). However, we briefly describe the main features which are common to all major software packages. According to Chapter 1, the description is organized along the project life cycle. Some features which can not be assigned to certain phases are presented at the end of the section. In the scientific literature, a few surveys on project management software and its properties can be found. Among these are Assad and Wasil (1986), Wasil and Assad (1988), De Wit and Herroelen (1990), Dworatschek and Hayek (1992), and Schneider and Hieber (1997). Hints on selecting project management software packages are, e.g., given in Kolisch and Hempel (1996 a).

Note that the description is restricted to project management software which supports integrated project planning and control in the stronger sense. Besides such software packages, a large variety of other packages covering only certain functions of project management exist. Examples are packages for configuration management or risk analysis. Furthermore, a large amount of software which has not been developed for project management purposes only can favorably be used in the field of project management (cf., e.g., Dworatschek and Hayek (1992, pp. 23)). This, e.g., includes word processing, spreadsheet and presentation packages as well as software supporting electronic communication. Among the most popular project management software packages are Computer Associates' Super Project, Microsoft's Project, Primavera's Project Planner and Scitor's Project Scheduler.

2.5.1 Features for Project Conception, Definition, and Termination

During the early phases of a project, i.e., the conception and the definition phase, the standard project management software can not explicitly support accomplishing the corresponding project management tasks. However, it may still provide a useful instrument. In the *conceptual phase*, it offers a possibility to comfortably access data on former projects within a short time and with little effort. Based on the experience contained in these data, cost, durations, resource requirements, and the like can be estimated more efficiently and accu-

rately. Having such information on hand is, e.g., important for performing a feasibility study and for selecting projects among a set of candidates.

Also in the *definition phase*, the major utility of project management software consists of having data on former projects available. If similar projects have already been executed earlier, the corresponding project plans, i.e., work breakdown structures, project networks, schedules, and budgets, may be exploited in order to derive a suitable process organization as well as for defining the budget. Furthermore, nearly all packages offer the possibility to work on different levels of detail, i.e., the major phases or milestones can be planned using features which are provided for the project planning phase in the main. By doing so, the stakeholders of the project can achieve an impression of important sub-activities and their desired completion times.

Like in the early phases of the project, the usability of project management software is restricted in the *completion phase*. Nevertheless, if the software is applied appropriately during the execution of the project, the data recorded again may provide valuable information which can be exploited when creating the final report. Furthermore, by storing the data of completed projects, an experience data base may successively be developed.

2.5.2 Features for Project Planning

Based on the specifications made in the former phases, the planning phase is concerned with preparing the execution of the project in detail. In general, the planning phase represents the phase in which the use of project management software packages is most promising. Therefore, the typical features supporting the accomplishment of the major management tasks in this phase are described in detail.

Structuring. As described in Section 2.1.1, the *work breakdown structure* represents the major tool for identifying the sub-activities and jobs the execution of which is required for terminating a project successfully. The development of this structure is greatly facilitated by project management software. For creating the multi-level work breakdown structure, sub-activities have successively to be broken into smaller ones. The level of the work breakdown structure on which a sub-activity is located is usually visualized by its presentation format, e.g., the chosen font size or style. Furthermore, codes identifying each sub-activity and job are automatically generated such that they reflect the structure de-

veloped so far as a hierarchical list. Finally, the work breakdown structure may be depicted graphically. Accordingly, most packages offer the possibility to define an organizational breakdown structure and to develop linear responsibility charts (cf. Section 1.5.1).

Having defined the jobs to be performed, the next step of structuring consists in setting forth their durations and introducing precedence relationships among them. Based on these data, the project is usually presented as an *activity-on-node network* or as an extended *Gantt chart*. In the latter case, a simple critical path analysis is performed automatically while entering the data. In general, modern software packages provide two different input possibilities. The first one is to use a table according to Table 2.1. The second one consists of graphically defining the data with a mouse as input device. For example in a Gantt chart representation, the duration of a job may be altered by shortening or extending the appropriate bar. Precedence relationships may be specified by drawing a line connecting two jobs. During the input, consistency tests are performed detecting, e.g., possible cycles in the project network. Note that most packages support the different types of precedence relationships defined by the precedence diagramming method but are not able to consider maximum time lags. Furthermore, for each job specific release and due dates can be defined at which it can be started the earliest or has to be finished the latest, respectively. Finally, it may be specified whether the execution of the job may be interrupted or not. This is in particular important within the context of resource-constrained scheduling.

Scheduling. Within most packages, scheduling starts with developing a *project calendar*. This calendar defines working periods as well as non-working ones caused by weekends, holidays, and vacations. The so-defined calendar is considered within schedule computations by assuming that the processing of jobs which would be in progress on non-working periods can be stopped before such a period and immediately be continued afterwards. For example, the execution of a job with a duration of 7 days may be started on a Monday. On Friday evening it is interrupted until Monday morning when it is resumed. Hence, the job is terminated on Tuesday evening.

Subsequently, a simple *critical path analysis* is performed considering release and due dates. As stated above, this is usually done automatically while entering the job data. Concurrently a project schedule is developed. By default, the software packages schedule the jobs such that their execution is started as soon

as possible, i.e., at the earliest starting time obtained by the critical path analysis, respectively. Alternatively, single jobs can be defined to start as late as possible, e.g., in order to postpone possible associated cash outflows. If the determined starting time of a job still does not correspond to the one desired by the user, it may be changed manually, e.g., by moving the appropriate bar in the Gantt chart representation. Slack time calculations usually restrict to computing the total float of each job. As a result of the critical path analysis, jobs which are critical are set off in a different color in the Gantt chart as well as in the network representation.

Resource Allocation. The process of resource allocation starts with defining the resource types to be considered. In general, each resource type may belong to a certain class, like, e.g., human resources or materials representing renewable and non-renewable resources, respectively. Subsequently, for each renewable resource type a calendar is developed based on the project's one describing its availability in each working period of the planning horizon. In some software packages, this availability is assumed to be constant over all working periods whereas in more evolved ones it is allowed to vary. In case that a resource type belongs to the class of human resources often relationships to the organizational breakdown structure can be introduced, i.e., for each resource type the corresponding organizational unit, e.g., the engineering team, is identified.

Having available the data of the resource types provided for executing the project, the respective requirements of the jobs have to be specified. Two basic approaches can be distinguished within project management software packages. In the first one, the total requirement for performing a job is determined. Then, its actual duration depends on the amount of resources which are allocated to it in each period of its execution. In the other, more common approach, the job has a fixed duration and a constant per period usage of the resource type. Only in a few packages, the resource usage of jobs is allowed to vary during their processing.

Finally, for the schedule obtained by the critical path analysis, the *resource loading profile* of each resource type is calculated and visualized in form of a histogram (cf. Section 2.3.1). In case of resource conflicts being detected, these can be resolved either manually or automatically. In order to support the first option, it is often possible to show a Gantt chart simultaneously on a split screen such that the jobs involved in conflicts can be identified. When resolving resource conflicts automatically, the *generalized resource-constrained project*

scheduling problem is solved heuristically (cf. Section 3.2.2 and Chapter 5). However, the capability of finding near optimal solutions may vary considerably depending on the type of heuristic implemented within certain software packages. Therefore, in recent publications, different software packages have been compared concerning the quality of the solutions determined for sets of test projects (cf. Johnson (1992), Maroto and Thomas (1994), Farid and Manoharan (1996), and Kolisch and Hempel (1996 b, c), Maroto et al. (1999) as well as Section 7.4.1.4, pp. 295).

Budgeting. Besides the planning functions described so far, most software packages additionally contain a simple budgeting function. For this purpose, cost for providing the different resource types have to be specified, respectively. In case of, e.g., a resource type belonging to the class of human resources or equipment, cost are usually given as per period cost. By the way of contrast, cost per unit are defined for materials. Some packages even offer the possibility to distinguish between normal cost and cost arising if capacity beyond the defined per period availability has to be provided, e.g., overtime cost. Furthermore, overhead cost incurring in each period of the project's execution may be included. When planning long term projects, additionally inflation plans may be considered. Such plans correspond to formulae describing the expected increase of cost due to inflation. Also when parts of the project are executed in different countries, exchange rates between currencies may be defined. Having collected all these data, the expected cost for realizing the current schedule is computed by the software package.

2.5.3 Features for Project Execution

Reporting. In order to conduct and control a project efficiently, the plans developed have to be communicated among participants of the project. For this purpose, modern software packages offer a large number of standardized reports. Additionally, the reports may be customized by the user such that they meet his individual demands. For example, they may be adapted to the terminology of the organization or to the corporate design.

First of all, the work breakdown structure, the organizational breakdown structure as well as the linear responsibility chart may be printed in a large number of different representations. For example, for each organizational unit, a list of jobs in the execution of which it is involved may be prepared. Accordingly, the

software packages are able to draw the project network after introducing the precedence relationships. This is a complex task, because in order to achieve a clear representation, intersections between arcs connecting nodes have to be minimized. Furthermore, with an increasing number of jobs, reasonable prints of the project networks can only be obtained by, e.g., plotters which are able to handle a large paper size. Therefore, many packages allow the use of Hammock jobs which replace a set of single jobs. This type of aggregation is appropriate for high-level reports, e.g., to the senior management.

Having developed a schedule, the user may choose among several Gantt chart representations. For example, precedence relationships may be included or not, critical activities may be set off in a different color, and total floats may be depicted. As already mentioned above, the loading of the different resource types may be visualized by histograms, respectively. Furthermore, for each resource type the jobs which require it may be listed separately.

Monitoring and Control. In order to control the performance of a project efficiently, its progress has to be monitored continually. Using project management software, this requires carefully and steadily collecting data on the advances in the processing of the active jobs. Within most packages, updating data can usually be performed in two different ways. The first possibility consists of updating the project up to a certain period. In this case, all jobs which have been scheduled to be finished until this period are considered to be completed and the data is modified correspondingly. However, this method can only be applied, if all the affected jobs are really terminated. Otherwise, the performance data on each job in progress has to be modified separately. According to entering job data during structuring, this can be done either textually by using a keyboard or graphically with a mouse. The resulting effort associated with keeping the project data up-to-date is often used as an argument for advocating against the application of project management software. Due to electronic mail becoming wide spread, this argument is no longer valid (cf. Helbrough (1995), Schönert (1998), Schott et al. (1998)). Relying on this technology, the updating process can be automated to a large extent. For this purpose, the software package identifies the organizational units responsible for the execution of the jobs in progress and generates corresponding electronic mails requesting the current status of termination. With the replies being structured according to a standard format, they may directly be evaluated by the software packages. Additionally, some software packages are able to estimate the expected completion dates of

active jobs being delayed. Having all data available, the project performance is usually depicted in form of a *progress Gantt chart* (cf. Section 2.4.1). According to user-defined filters, certain jobs may be set off by coloring or shading, e.g., when their total float in percent of their duration falls below a certain level or when their processing has been postponed several times. Finally, a few packages are able to generate milestone trend analysis diagrams.

Besides monitoring and controlling the on schedule performance of projects, most software packages are also able to track the cost performance. For a given period, they can compute the budgeted cost of work scheduled. Accordingly, with data on the progress of jobs becoming available during the project execution, the budgeted cost of work performed are calculated automatically. Furthermore, additional cost arising for a job may be entered which have to be considered when determining the actual cost of work performed. Having provided these measures, an *earned value analysis* can be accomplished. For this purpose, the cost and schedule variance is computed and the cost curve is depicted graphically (cf. Section 2.4.2).

2.5.4 General Features

When planning the execution of the project, usually several alternatives for its realization have to be examined by the project manager. Furthermore, *what-if analyses* have to be performed in order to investigate the effects of processing the project under different conditions. For example, the project performance assuming different resource availabilities may be evaluated. These tasks are considerably facilitated when project management software is used. First of all, the user is able to create several baseline plans, i.e., schedules and budgets which can be stored and compared automatically. While developing these plans, the user will need to modify the input data as well as the proposed schedules. Since usually several tries are necessary for finding the modifications resulting in a satisfying schedule, many software packages offer "undo" functions allowing to take back some of the most recent modifications.

If within a parent organization several projects are executed concurrently, they may eventually use the same set of resources. In this case, resource conflicts have to be avoided taking into account the requirements of all projects. For this purpose, many software packages offer the possibility to perform a multi-project planning according to one of the following approaches. Within the first

approach, several projects may be combined to a single "master-project". After allocating resources within this master-project, it is broken into the original ones again. Another approach consists of simultaneously loading the data of several projects into the computer's main storage and comparing their resource loadings. However, this approach is limited when large projects have to be co-ordinated because it depends on the availability of computer memory. The most convenient approach is based on a central resource data base. When planning a single project, all data affecting the availabilities of resources are accessed and stored in this data base such that resource conflicts can be detected immediately.

Furthermore, some software packages can be run within a computer network. In this case, several users are allowed to work simultaneously even on one and the same project. Therefore, according mechanisms have to be provided which avoid that inconsistencies in the project data occur. This may, e.g., be achieved by granting the rights for certain operations only to a subset of users. For example, adapting the project schedule may only be allowed to the project manager. Furthermore, other members of the project team may have restricted access, e.g., only to the data of jobs in the execution of which they are involved.

Finally, modern software packages provide the possibility of comfortably including own procedures, e.g., for resource-constrained scheduling, by incorporated programming languages (e.g., Visual Basic for Applications which is a part of Microsoft Project). An important development in this context is that the software packages may exchange data with other software applications using relational data bases. This, e.g., allows automatically incorporating data from a cost accounting system (cf. Heindel and Kasten (1996)).

3 Resource-Constrained Scheduling Problems

This chapter is devoted to the description of mathematical optimization problems which arise in the context of resource-constrained project scheduling. In Section 3.1, some notations and definitions are introduced which are used throughout the forthcoming chapters. Section 3.2 presents different mathematical formulations for the fundamental resource-constrained project scheduling problem (RCPSP). Furthermore, a practically important generalization for which appropriate solution methods are described in the Chapters 4 to 6 is introduced. In Section 3.3, further possible extensions of RCPSP are examined. Finally, closely related scheduling problems are surveyed in Section 3.4. Since solution procedures for the problems presented in the latter two sections are not part of this book, references on corresponding literature are provided, respectively. Furthermore, for each reference the basic type of the solution procedure proposed, e.g., priority-rule based heuristic, tabu search or branch and bound, is stated. For brief introductions into such solution approaches, we refer to the Chapters 5 and 6. General surveys on procedures for resource allocation are contained in Davis (1966), Laue (1968), Herroelen (1972), Davis (1973), Slowinski (1977), Ritchie (1985), Domschke and Drexl (1991), Icmeli et al. (1993), Elmaghraby (1995), Özdamar and Ulusoy (1995), Herroelen et al. (1996), Kolisch and Padman (1997), and Brucker et al. (1999). Brucker et al. (1999) and Herroelen et al. (1999) both describe possible classification schemes for resource-constrained project scheduling problems.

3.1 Notations and Definitions

In the following, we introduce some definitions required for a proper statement of the resource-constrained project scheduling problems to be discussed. Some of them have already been provided in Chapter 2. However, they are repeated in a more formal way for presentation purposes. Table 3.1 shows a summary of the notations and terms most frequently used in this chapter.

Table 3.1. Notations and Definitions

Notation	Definition
n	number of jobs
J	set of all jobs; $J = \{1, \dots, n\}$
j	index for the jobs; $j = 1, \dots, n$
d_j	duration of job j in periods ^a
rd_j	release date of job j
dd_j	due date of job j
P_j / F_j	set of jobs which immediately precede / follow job j
P_j^* / F_j^*	set of jobs which precede / follow job j
λ_{ij}	start-to-start minimum time lag between two jobs i and j in number of periods ^b
\bar{T}	end of the planning horizon
t	index for periods; $t = 1, \dots, \bar{T}$
ES_j	earliest starting time of job j
LS_j	latest starting time of job j
EF_j	earliest finishing time of job j
LF_j	latest finishing time of job j
$E(t)$	jobs which may be processed in period t ($E(t) = \{j \mid j \in J \text{ and } ES_j + 1 \leq t \leq LF_j\}$)
m	number of renewable resource types
R	set of all renewable resource types; $R = \{1, \dots, m\}$
r	index for the renewable resource types; $r = 1, \dots, m$
a_r	constant per period availability of resource type r
a_{rt}	availability of resource type r in period t
u_{jr}	per period resource usage of resource type r by job j

- a. Job durations are assumed to be integral. This can be done without a relevant loss of generality, because non-integer parameter values of real-word problems can be converted into integer ones by applying appropriate scaling operations.
- b. In Section 2.1.2, start-to-start minimum time lags have been abbreviated by λ_{ij}^{SS} to distinguish them from the other possible types. Since in this and the forthcoming chapters only this type of minimum time lag is considered, we use λ_{ij} from now on.

Definition 3.1. A *finish-to-start project network* $G = (J, A, \mathbf{d}, \mathbf{u})$ is a non-cyclical digraph (activity-on-node network) with a set $J = \{1, \dots, n\}$ of n nodes and a set of arcs $A = \{(i, j) \mid j \in J \text{ and } i \in P_j\}$. The nodes depict jobs of a project, whereas the arcs represent finish-to-start precedence relationships. A job i which must be completed before a job j can be started is called a *predecessor* of i , while j is a *follower* or *successor* of i . A precedence relationship between a pair of jobs (i, j) is called *direct* or *immediate* if no follower of job i is also a predecessor of j . The set of immediate predecessors (successors) of job j form the set P_j (F_j). To ease presentation, the nodes are numbered following a *topological ordering*, i.e., the inequality $i < j$ is fulfilled for each arc $(i, j) \in A$. The node weights d_j denote the integral durations of the jobs. Furthermore, the vector $\mathbf{u}_j = u_{j1}, \dots, u_{jm}$ defines the resource usages of the renewable resource types $r = 1, \dots, m$ which are required in each period of executing a job j .

Definition 3.2. According to a finish-to-start project network, a *start-to-start project network* $G = (J, A, \lambda, \mathbf{d}, \mathbf{u})$ is a non-cyclical digraph. However, each arc (i, j) represents a direct start-to-start precedence relationship. The corresponding arc weight $\lambda_{ij} \geq 0$ denotes a minimum time lag which has to be observed between the start of job i and the start of job j .

Definition 3.3. A node representing a job without predecessors (followers) is termed a *source (sink)* of G . To ease presentation, the jobs 1 and n are assumed to be the only source and sink of G , respectively. Furthermore, their durations are assumed to be $d_1 = d_n = 0$.

Definition 3.4. A sequence $\alpha_1, \alpha_2, \dots, \alpha_q$ of arcs $\alpha_h \in A$ is called a *path* π of G , if a sequence of nodes $\langle j_0, j_1, \dots, j_q \rangle$ exists such that $\alpha_h = (j_{h-1}, j_h)$ for $h = 1, \dots, q$. In a finish-to-start project network, the *length* $l(\pi)$ of a path π is defined by the number of periods which due to this path have to pass away between finishing job j_0 and starting job j_h . It can be calculated as follows:

$$l(\pi) = \sum_{h=1}^{q-1} d_{j_h} \quad (3.1)$$

In a start-to-start project network, the length of the path π corresponds to the number of periods which have to pass away between the start of job j_0 and j_h considering π . It is computed by:

$$l(\pi) = \sum_{h=0}^{q-1} \lambda_{j_h, j_{h+1}} \quad (3.2)$$

Definition 3.5. The *transitive closure* $G^* = (J, A^*, \mathbf{d}, \mathbf{u})$ of a finish-to-start project network G also contains arcs (i, j) for all pairs of jobs i and j which are connected by a path in G . That is, $A^* = \{(i, j) \mid j \in J \text{ and } i \in P_j^*\}$ with P_j^* denoting the set of all predecessors of a job j . Those arcs included in the difference set $A^* - A$ are called *transitive* and represent *indirect precedence relationships*. If transitive arcs are already contained in G , they are *redundant* and can be omitted.

The transitive closure $G^* = (J, A^*, \lambda, \mathbf{d}, \mathbf{u})$ of a start-to-start project network is defined accordingly. However, within a start-to-start network not all transitive arcs contained are redundant. If a pair of jobs (i, j) is connected by an arc as well as by a number of paths, this arc must not be omitted when the corresponding minimum time lag λ_{ij} exceeds the length of the longest path between the jobs i and j containing more than one node.

Definition 3.6. In case of constant resource availability during the planning horizon, a pair of jobs (i, j) is said to be *incompatible* if both jobs can not be processed in parallel. This is true, if they are either related by precedence, i.e., $(i, j) \in A^*$ or $(j, i) \in A^*$, or their cumulated per period capacity requirement $u_{ir} + u_{jr}$ exceeds the availability a_r for any resource type $r = 1, \dots, m$. The collection of all incompatible pairs is denoted by IP .

In general, a set S of jobs is incompatible, if it contains at least one pair $(i, j) \in IP$ or if its total resource demand $\sum_{j \in S} u_{jr}$ is larger than the per period availability a_r for any resource type $r = 1, \dots, m$. A *resource incompatible set* S is present, if only the last condition is fulfilled. It is called *minimal*, if the incompatibility is resolved after arbitrarily removing a single job. The collection of all minimal resource incompatible sets is given by IS .

3.2 Basic Models

In this section, we present different mathematical formulations for the well-known resource-constrained project scheduling problem (RCPSP) which has extensively been studied in the literature during the last three decades. Furthermore, we describe a generalized version which allows for modeling real-world scheduling problems more realistically.

3.2.1 The Resource-Constrained Project Scheduling Problem (RCPSP)

The RCPSP can be described as follows (for a more comprehensive discussion as well as an example cf. Section 2.3.2). A *project*, given by a *finish-to-start network*, has to be realized. *No preemption* of jobs is allowed, i.e., whenever a job has been started at a time point t it must be performed consecutively during the periods $t+1, \dots, t+d_j$ (for the difference between time points and periods see Remark 3.2). When performing the project, *resource constraints* have to be observed. The basic type of RCPSP considers m (renewable) resource types, defined by the index set $R = \{1, \dots, m\}$. In each period of the planning horizon, a type r is available in a *constant number* a_r of units ($r = 1, \dots, m$).

The RCPSP is to find a non-preemptive schedule of the jobs, i.e., a set of starting times, such that the precedence and resource constraints are satisfied and the project duration is minimized (cf. Definition 5.1., p. 163). Note that $u_{jr} \leq a_r$ for $j = 1, \dots, n$ and $r = 1, \dots, m$ represents a necessary condition for existence of a feasible schedule.

In the literature, RCPSP has extensively been studied. This is due to the fact that a large number of possible applications beyond project scheduling exist in the field of production management. For example within CIM (Computer Integrated Manufacturing), similar problems arise in the design of Leitstand systems (cf. Drexel and Kolisch (1993 b), Drexel et al. (1994 a), Drexel and Kolisch (1996)). Furthermore, RCPSP is well-suited for capacity planning in case of make-to-order production (cf. Drexel and Kolisch (1993 a), Drexel et al. (1994 b), and Franck et al. (1997)). Finally, other fundamental production planning problems, such as the flow shop and the job shop scheduling problem, may be modeled as resource-constrained project scheduling problems (cf., e.g., Schrage (1970), Baker (1974), Drexel (1990), Sprecher (1994, chapter 2)). As a result, a number of attempts have been made to develop mathematical formulations for modeling RCPSP. In the following, after defining some basic properties of RCPSP, we present the most important ones as well as introduce new ones. A formulation which also allows for non-integer durations of jobs is given by Icmeli and Rom (1996).

3.2.1.1 Properties of RCPSP

In order to develop efficient mathematical optimization models describing RCPSP, the number of variables and constraints to be considered has to be kept

as small as possible. This can be achieved by restricting the number of periods in which a job may potentially be processed. For this purpose, an upper bound \bar{T} on the optimal project completion time has to be calculated. This can, e.g., be done by applying one of the heuristic procedures described in Chapter 5. However, if such procedures are not available, a simple upper bound can be computed by:

$$\bar{T} = \sum_{j=1}^n d_j \quad (3.3)$$

Based on this upper bound, earliest and latest starting and finishing times can be obtained by a forward and a backward pass, respectively, as already shown in Section 2.2.1. Starting with $ES_1 = EF_1 = 0$, the forward pass calculates earliest starting and finishing times as follows:

$$ES_j = \max\{EF_i \mid i \in P_j\}; EF_j = ES_j + d_j \quad \text{for } j = 2, \dots, n \quad (3.4)$$

The backward pass is performed beginning with $LF_n = LS_n = \bar{T}$. This yields latest finishing and starting times LF_j and LS_j as follows:

$$LF_j = \min\{LS_h \mid h \in F_j\}; LS_j = LF_j - d_j \quad \text{for } j = n-1, \dots, 1 \quad (3.5)$$

Remark 3.1. The interval $[ES_j, LF_j]$ defines a *time window* in which job j has to be processed when the project is to be completed after \bar{T} periods. That is, if the project completion time is limited to a maximum number \bar{T} of periods, this has not to be modeled explicitly by a corresponding restriction but can be incorporated by calculating the time windows accordingly. If the time windows are only used for verifying that no better solution than with a project completion time of \bar{T} periods exists, the backward pass can be initialized by $\bar{T}-1$.

Remark 3.2. The earliest and latest starting and finishing times correspond to time points delimiting periods. Since the difference between the terms "time point" and "period" is important for the understanding of the forthcoming expositions, it is graphically depicted in Figure 3.1. The figure shows, that two time points t and $t+1$ define the start and the end of period $t+1$, respectively. That is, if a job j is scheduled to start at time point t and to finish at time point $t+d_j$ it will be processed in the periods $t+1, \dots, t+d_j$. Furthermore, if its earliest starting time (point) is ES_j , the earliest period in which it can be processed is ES_j+1 . However, with a latest finishing time (point) LF_j , the last period in which it can be executed is LF_j .

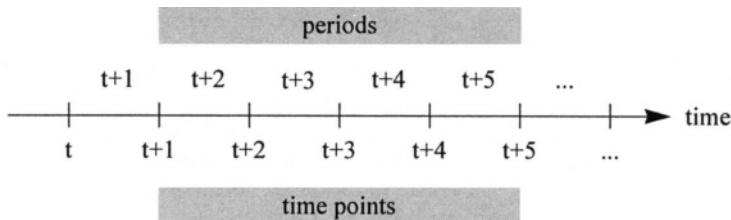


Figure 3.1. Difference between time points and periods

Remark 3.3. In order to examine whether a resource conflict occurs in a period t , only those jobs have to be considered which can really be processed in this period. They form the set of jobs $E(t) = \{j \mid j \in J \text{ and } ES_j + 1 \leq t \leq LF_j\}$ which are *eligible* for a period t .

3.2.1.2 Formulation 1

One of the first mathematical formulations, explicitly developed for RCPSP, has been presented by Pritsker et al. (1969). In this formulation, solutions are represented by 0-1 variables x_{jt} which are defined as follows:

$$x_{jt} = \begin{cases} 1, & \text{if job } j \text{ is finished at the end of period } t \\ 0, & \text{otherwise} \end{cases} \quad \text{for } j \in J \text{ and } t \in [EF_j, LF_j] \quad (3.6)$$

Note that the variables x_{jt} of a job must only be introduced in those periods at the end of which it may be terminated. For a given solution vector of an RCPSP instance, the period SF_j in which a job j is scheduled to be finished is determined by:

$$SF_j = \sum_{t=EF_j}^{LF_j} t \cdot x_{jt} \quad (3.7)$$

By means of the above parameters and variables, RCPSP can now be formulated by the objective function (3.8) and the constraints (3.9) - (3.12).

$$\text{Minimize } CT(\mathbf{x}) = \sum_{t=EF_n}^{LF_n} t \cdot x_{nt} \quad (3.8)$$

subject to

$$\sum_{\substack{t=EF_j \\ t=1}}^{LF_j} x_{jt} = 1 \quad \text{for } j \in J \quad (3.9)$$

$$\sum_{\substack{t=EF_j \\ t=1}}^{LF_j} (t-d_j) \cdot x_{jt} - \sum_{\substack{t=EF_i \\ t=1}}^{LF_i} t \cdot x_{it} \geq 0 \quad \text{for } j \in J \text{ and } i \in P_j \quad (3.10)$$

$$\sum_{j \in E(t)} u_{jr} \cdot \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_j-1, LF_j\}} x_{jq} \leq a_r \quad \text{for } r \in R \text{ and } t \in [1, \dots, \bar{T}] \quad (3.11)$$

$$x_{jt} \in \{0, 1\} \quad \text{for } j \in J \text{ and } t \in [EF_j, LF_j] \quad (3.12)$$

The objective function (3.8) minimizes the completion time of the dummy end job n. The assignment constraints (3.9) together with the integrality constraints (3.12) guarantee that exactly one finishing period is assigned to each job j. In order to observe the precedence constraints, no job must be started before all its predecessors have been completed which is ensured by the restrictions (3.10). Finally, resource constraints (3.11) have to be fulfilled for each period $t \in [1, \dots, \bar{T}]$ and each resource type $r \in R$. For this purpose, the total resource demand of the jobs $j \in E(t)$ which are actually processed in period t has to be determined. A job j is executed in a period t, if it is finished at the end of the periods $t, \dots, t+d_j-1$, respectively. However, it must be terminated within the time window $[EF_j, LF_j]$.

The formulation (3.8)-(3.12) requires the definition of up to $n \cdot \bar{T}$ binary variables and consists of up to $n + |A| + m \cdot \bar{T}$ restrictions. Because of $|A|$ containing at most $n \cdot (n-1)/2$ arcs, the model requires $O(n^2 + m \cdot \bar{T})$ restrictions. The O-notation which denotes the order of a function is introduced in Section 3.2.3.

3.2.1.3 Formulation 2

The following formulation has initially been introduced by Kaplan (1988) for the resource-constrained project scheduling problem with preemption as well as by Neumann and Morlock (1993, section 3.7) for the resource leveling problem (cf. Sections 3.3.1 and 3.4.2, respectively). In the following, it is adapted to RCPSP. It is based on defining 0-1 variables, which indicate whether a job j is active in period t or not:

$$x_{jt} = \begin{cases} 1, & \text{if job } j \text{ is processed in period } t \\ 0, & \text{otherwise} \end{cases} \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, LF_j] \quad (3.13)$$

Note that this formulation requires job durations larger than zero because otherwise the fulfillment of the precedence constraints can not be verified. However, this does not restrict its applicability. The dummy start and end job are assigned durations $d_1 = 1$ and $d_n = 1$ which have already to be considered when performing the forward and the backward pass, respectively. All other jobs i with a duration of $d_i = 0$ can be eliminated from the project network. In order to preserve transitive precedence relationships, an arc (h, j) is included for each pair of jobs with $h \in P_i$ and $j \in F_i$. The solution obtained for the modified problem instance may be transferred into one for the original one by decreasing the starting times of all jobs by one period (the duration of the dummy start job). With the binary variables x_{jt} , RCPSP can now be formulated as follows:

$$\text{Minimize } CT(\mathbf{x}) = \sum_{t=ES_n+1}^{LF_n} t \cdot x_{nt} \quad (3.14)$$

subject to

$$\sum_{t=ES_j+1}^{LF_j} x_{jt} = d_j \quad \text{for } j \in J \quad (3.15)$$

$$d_j \cdot (x_{jt} - x_{j,t+1}) - \sum_{q=ES_j+1}^t x_{jt} \leq 0 \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, LF_j - 1] \quad (3.16)$$

$$d_i \cdot x_{jt} - \sum_{q=ES_i+1}^{t-1} x_{it} \leq 0 \quad \text{for } j \in J, i \in P_j, t \in [ES_j + 1, \dots, LF_i] \quad (3.17)$$

$$\sum_{j \in E(t)} u_{jr} \cdot x_{jt} \leq a_r \quad \text{for } r \in R \text{ and } t \in [1, \dots, \bar{T}] \quad (3.18)$$

$$x_{jt} \in \{0, 1\} \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, LF_j] \quad (3.19)$$

The objective function (3.14) minimizes the project completion time, because the given sum yields the smallest possible value if the dummy end job n is completed as early as possible. The duration constraints (3.15) ensure that each job j is executed for d_j periods. Furthermore, to guarantee that the processing of each job is not interrupted, the non-preemption constraints (3.16) are introduced. A job j is only allowed to be executed in a period t but not in the subsequent period $t+1$, if it is terminated at the end of period t . This is the case, when it has already been processed for d_j periods. The precedence constraints (3.17) follow the same idea. A job j must not be started before all its predecessors $i \in P_j$ have been processed completely, i.e., for d_i periods. The restrictions (3.18) represent the resource constraints.

The formulation (3.14)-(3.19) considers up to $n \cdot \bar{T}$ binary variables and up to $n + n \cdot \bar{T} + |A| \cdot \bar{T} + m \cdot \bar{T}$ restrictions. Since $|A|$ contains at most $n \cdot (n-1)/2$ arcs and $m < n$ for reasonable problem instances, it requires $O(n^2 \cdot \bar{T})$ restrictions which exceeds the number required by formulation 1 by far.

3.2.1.4 Formulation 3

This new formulation uses 0-1 variables y_{jt} which are defined as follows:

$$y_{jt} = \begin{cases} 1, & \text{if job } j \text{ starts at the beginning of period } t \text{ or earlier} \\ 0, & \text{otherwise} \end{cases} \quad \text{for } j \in J \text{ and } t \in [ES_j + 1 - d_j, LF_j] \quad (3.20)$$

Within a feasible solution obtained for the following model (3.22)-(3.28), each vector $(y_{j,ES_j+1-d_j}, \dots, y_{j,LF_j})$ consists of a consecutive block of zeros followed by a block of ones. The changeover between the two blocks takes place with the element y_{jt} representing the period at the beginning of which job j is started. Hence, assuming that all jobs $j \in J$ have a duration $d_j > 0$ like in formulation 2, the scheduled starting time point SS_j of a job j can be calculated by:

$$SS_j = LF_j - \sum_{t=ES_j+1}^{LF_j} y_{jt} \quad (3.21)$$

In the literature, this way of defining variables has already been applied for formulating multi-level lotsizing problems (cf., e.g., McKnew et al. (1991) and

Domschke et al. (1997, pp. 156)) as well as assembly line balancing problems (cf. Scholl (1999, section 2.2.1.2)). With this definition of binary variables, RCPSP can be formulated as follows:

$$\text{Minimize } CT(y) = \bar{T} - \sum_{t=ES_n+1}^{LS_n} y_{nt} \quad (3.22)$$

subject to

$$y_{j,t+1} - y_{jt} \geq 0 \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, LS_j - 1] \quad (3.23)$$

$$y_{i,t-d_i} - y_{jt} \geq 0 \quad \text{for } j \in J, i \in P_j, t \in [ES_j + 1, \dots, LF_i] \quad (3.24)$$

$$\sum_{j \in E(t)} u_{jr} \cdot (y_{jt} - y_{j,t-d_j}) \leq a_r \quad \text{for } r \in R \text{ and } t \in [1, \dots, \bar{T}] \quad (3.25)$$

$$y_{jt} = 0 \quad \text{for } j \in J \text{ and } t \in [ES_j + 1 - d_j, ES_j] \quad (3.26)$$

$$y_{jt} = 1 \quad \text{for } j \in J \text{ and } t \in [LS_j + 1, LF_j] \quad (3.27)$$

$$y_{jt} \in \{0, 1\} \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, LS_j] \quad (3.28)$$

The smallest possible project completion time is determined by using the objective function (3.22). Together with (3.26)-(3.28), the assignment constraints (3.23) ensure that each job j is exactly assigned one starting period. The precedence constraints are represented by the restrictions (3.24). If a job j is to be processed in a period t , all its predecessors $i \in P_j$ must have been active at least d_i periods earlier. Within the resource constraints (3.25) all jobs which are executed in a period t have to be considered. For a job j being started in period t or earlier ($y_{jt} = 1$), this is true, if it has not been in progress $t - d_j$ periods before ($y_{j,t-d_j} = 0$).

Remark 3.4. The restrictions (3.26) and (3.27) can be eliminated from the model by replacing the respective variables within the other constraints. The same possibility exists for some of the restrictions in the formulations 4 and 5, respectively.

The model (3.22)-(3.28) consists of up to $n \cdot \bar{T}$ binary variables. The number of restrictions to be introduced is at most $n \cdot \bar{T} + |A| \cdot \bar{T} + m \cdot \bar{T}$. Following the argumentation for formulation 2 results in an order of $O(n^2 \cdot \bar{T})$ restrictions which is

due to the precedence restrictions. Though this exceeds the number required by formulation 1, formulation 3 has the advantage that in all restrictions except for (3.25) only coefficients 1, -1, and 0 occur, respectively. This may, e.g., be advantageous if the resource constraints are relaxed using Lagrangean multipliers in order to calculate a lower bound on the project completion time (cf. Section 4.1.2.2). However, in order to reduce the number of restrictions, (3.24) may be reformulated using (3.21) reducing the number of restrictions to the order of $O(n^2 + m \cdot \bar{T})$ as in formulation 1:

$$\left(LF_j - \sum_{t=ES_j+1}^{LF_j} y_{jt} \right) - \left(LF_i - \sum_{t=ES_i+1}^{LF_i} y_{it} \right) \geq d_i \quad \text{for } j \in J \text{ and } i \in P_j \quad (3.29)$$

3.2.1.5 Formulation 4

We modify formulation 3 by redefining the range of the 0-1 variables y_{jt} and additionally introducing the variables z_{jt} as follows:

$$y_{jt} = \begin{cases} 1, & \text{if job } j \text{ starts at the beginning of period } t \text{ or earlier} \\ 0, & \text{otherwise} \end{cases} \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, LF_j] \quad (3.30)$$

$$z_{jt} = \begin{cases} 1, & \text{if job } j \text{ is terminated at the end of period } t \text{ or later} \\ 0, & \text{otherwise} \end{cases} \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, LF_j] \quad (3.31)$$

In contrast to the possible solution vector $(y_{j,ES_j}, \dots, y_{j,LF_j}), (z_{j,ES_j}, \dots, z_{j,LF_j})$ starts with a consecutive block of ones and is terminated by a block of zeros. With these definitions, we yield the following formulation (3.32)-(3.41):

$$\text{Minimize } CT(y, z) = EF_n + \sum_{t=EF_n+1}^{LF_n} z_{nt} \quad (3.32)$$

subject to

$$y_{j,t+1} - y_{jt} \geq 0 \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, LS_j - 1] \quad (3.33)$$

$$z_{jt} - z_{j,t+1} \geq 0 \quad \text{for } j \in J \text{ and } t \in [EF_j + 1, \dots, LF_j - 1] \quad (3.34)$$

$$\sum_{t=ES_j+1}^{LF_j} y_{jt} + z_{jt} - 1 = d_j \quad \text{for } j \in J \quad (3.35)$$

$$y_{jt} + z_{it} \leq 1 \quad \text{for } j \in J, i \in P_j, t \in [ES_j + 1, \dots, LF_i] \quad (3.36)$$

$$\sum_{j \in E(t)} u_{jr} \cdot (y_{jt} + z_{jt} - 1) \leq a_r \quad \text{for } r \in R \text{ and } t \in [1, \dots, \bar{T}] \quad (3.37)$$

$$y_{jt} = 1 \quad \text{for } j \in J \text{ and } t \in [LS_j + 1, \dots, LF_j] \quad (3.38)$$

$$z_{jt} = 1 \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, EF_j] \quad (3.39)$$

$$y_{jt} \in \{0, 1\} \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, LS_j] \quad (3.40)$$

$$z_{jt} \in \{0, 1\} \quad \text{for } j \in J \text{ and } t \in [EF_j + 1, \dots, LF_j] \quad (3.41)$$

The minimal project completion time is determined by the objective function (3.32) which minimizes the number of consecutive ones contained in the vector $z_n = (z_{n,EF_n}, \dots, z_{n,LF_n})$. In addition with (3.38)-(3.41), the restrictions (3.33) and (3.34) warrant that exactly one starting and finishing period is determined for each job, respectively. Due to the restrictions (3.35), each job must be processed for exactly d_j consecutive periods. The restrictions (3.36) guarantee that the execution of a job j can not be started in a period t ($y_{jt} = 1$) if one of its predecessors $i \in P_j$ has not been terminated yet ($z_{it} = 1$). The resource constraints can be formulated by the restrictions (3.37). A job j is processed in a period t if it starts at the beginning of t or earlier ($y_{jt} = 1$) and finishes at the end of t or later ($z_{jt} = 1$). Concerning the possible elimination of the restrictions (3.38) and (3.39) see Remark 3.4.

The model (3.32)-(3.41) requires the definition of up to $2 \cdot n \cdot \bar{T}$ 0-1 variables. The number of restrictions to be introduced is at most $2 \cdot n \cdot \bar{T} + n + |A| \cdot \bar{T} + m \cdot \bar{T}$ and, hence, of the order $O(n^2 \cdot \bar{T})$. Of course, like with formulation 3, this order can be reduced to $O(n^2 + m \cdot \bar{T})$ by replacing the restrictions (3.36) by the ones (3.29). In comparison to all formulations presented so far, formulation 4 as it is has the advantage that the durations of jobs are not contained in the indices of variables or in the limits of sums. Thus, this model can also be applied when the

durations of jobs are part of the objective function. For example, for a given deadline on the project completion time, it may be desirable to determine a schedule for which the duration of a job is maximized. This may, e.g., be useful if the job is probably subject to delay. Furthermore, it may be helpful for determining critical jobs. If for the given project deadline the maximum duration of a job equals the estimated one, the job is critical.

3.2.1.6 Formulation 5

In the following, we present a mixed-integer formulation of RCPSP which has been developed by Alvarez-Valdés and Tamarit (1993). It is based on the following idea. Initially, the collection IS of all minimal resource incompatible sets is determined (cf. Definition 3.6., p. 76). In order to avoid a resource conflict which would be caused by concurrently processing the jobs of such a set S, it is sufficient to introduce a direct or a transitive precedence relationship between a pair of jobs $(i, j) \in S$. As a result, either job i has to be completely processed before job j can be started, or vice versa, and the corresponding resource conflict can no longer occur. This leads to the following definition of 0-1 variables. Note that $v_{ij} = 0$ does not imply that job j has to be performed before job i. This is indicated by $v_{ji} = 1$.

$$v_{ij} = \begin{cases} 1, & \text{if job } i \text{ is terminated before job } j \text{ is started} \\ 0, & \text{otherwise} \end{cases} \quad \text{for } i, j \in J \quad (3.42)$$

Furthermore, the formulation includes integer variables f_j denoting the period at the end of which a job j is finished. With these variables, RCPSP can be modeled as follows:

$$\text{Minimize } f_n \quad (3.43)$$

subject to

$$v_{ij} = 1 ; v_{ji} = 0 \quad \text{for } (i, j) \in A^* \quad (3.44)$$

$$v_{ij} + v_{ji} \leq 1 \quad \text{for } i, j \in J \text{ and } i \neq j \quad (3.45)$$

$$v_{ij} + v_{jk} - v_{ik} \leq 1 \quad \text{for } i, j, k \in J \text{ and } i \neq j, j \neq k, i \neq k \quad (3.46)$$

$$\sum_{\substack{i,j \in S \text{ and } i \neq j}} v_{ij} \geq 1 \quad \text{for all } S \in IS \text{ and } i \neq j \quad (3.47)$$

$$f_j - f_i \geq v_{ij} \cdot (d_i + M) - M \quad \text{for } i, j \in J \text{ and } i \neq j \quad (3.48)$$

$$v_{ij} \in \{0, 1\} \quad \text{for } i, j \in J \text{ and } i \neq j \quad (3.49)$$

$$f_j \in [EF_j, LF_j] \text{ and integer} \quad \text{for } j \in J \quad (3.50)$$

The finishing time of the dummy end job n is minimized by the objective function (3.43). The precedence relationships originally contained within the project network are represented by the definitions (3.44). Following Remark 3.4, p. 83, these restrictions can be eliminated by substitution. The restrictions (3.45) ensure that between two jobs at most one precedence relationship is introduced. Furthermore, the restrictions (3.46) guarantee that transitive precedence relationships are valid. The resource constraints are observed by introducing at least one precedence relationship between a pair of jobs for each incompatible set $S \in IS$ which is controlled by the restrictions (3.47). Finally, the restrictions (3.48) warrant that the values f_j are determined according to the original as well as to the artificial precedence relationships. M denotes a sufficiently large number. If a precedence relationship exists between two jobs i and j , the right-hand side of the restriction yields the value d_i . Otherwise, it equals $-M$ and, hence, there is no restriction of $f_j - f_i$.

The formulation (3.43)-(3.50) requires exactly n integer variables and up to n^2 binary variables. The number of restrictions to be introduced mainly depends on the number of minimal resource incompatible sets determined for a problem instance. In general, the number of such sets is of the order $O(2^{n-2})$, because the dummy jobs may not be included in any minimal resource incompatible set. Therefore, the applicability of this formulation is restricted to those problem instances where the actual number of such sets is low. The remaining constraints (3.45), (3.46), and (3.48) lead to the definition of $O(n^3)$ restrictions.

3.2.1.7 Formulation 6

This formulation has been introduced by Mingozi et al. (1998) in order to efficiently calculate lower bounds on the project completion time (cf. Section 4.1.2.3). In contrast to formulation 5, it is based on initially computing all *compatible sets* S_h ($h = 1, \dots, q$) of jobs which may be processed in parallel with-

out violating precedence or resource constraints. The collection of these sets is denoted by CS. Then, a solution of a problem instance with a project duration CT can be represented by a sequence of sets $\langle S_{h_1}, \dots, S_{h_t}, \dots, S_{h_{CT}} \rangle$, i.e., by defining the set which has to be processed in each period $t = 1, \dots, CT$, respectively. Accordingly, 0-1 variables p_{ht} are defined as follows:

$$p_{ht} = \begin{cases} 1, & \text{if set } h \text{ is processed in period } t \\ 0, & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{for } h = 1, \dots, q \text{ and} \\ t = 1, \dots, \bar{T} \end{array} \quad (3.51)$$

Furthermore, 0-1 variables x_{jt} according to (3.6), p. 79, have to be introduced indicating the period t at the end of which job j is terminated. The collection of compatible sets in which a job j is contained is denoted by $CS_j = \{S \mid S \in CS \text{ and } j \in S\}$. With these parameters and variables, RCPSP can be formulated as follows:

$$\text{Minimize } CT(\mathbf{x}, \mathbf{p}) = \sum_{\substack{t=EF_n \\ t=ES_n}}^{LF_n} t \cdot x_{nt} \quad (3.52)$$

subject to

$$\sum_{h=1}^q p_{ht} \leq 1 \quad \text{for } t \in [1, \dots, \bar{T}] \quad (3.53)$$

$$\sum_{h \in CS_j} \sum_{t=ES_j+1}^{LF_j} p_{ht} = d_j \quad \text{for } j \in J \quad (3.54)$$

$$\sum_{h \in CS_j} (p_{ht} - p_{h,t+1}) \leq 0 \quad \text{for } j \in J \text{ and } t \in [ES_j + 1, \dots, EF_j - 1] \quad (3.55)$$

$$\sum_{h \in CS_j} (p_{ht} - p_{h,t+1}) \leq x_{jt} \quad \text{for } j \in J \text{ and } t \in [EF_j, \dots, LF_j] \quad (3.56)$$

$$\sum_{t=EF_j}^{LF_j} x_{jt} = 1 \quad \text{for } j \in J \quad (3.57)$$

$$\sum_{\substack{t=EF_j \\ t=EF_i}}^{LF_j} (t-d_j) \cdot x_{jt} - \sum_{\substack{t=EF_i \\ t=EF_j}}^{LF_i} t \cdot x_{it} \geq 0 \quad \text{for } j \in J \text{ and } i \in P_j \quad (3.58)$$

$$x_{jt} \in \{0, 1\} \quad \text{for } j \in J \text{ and } t \in [EF_j, \dots, LF_j] \quad (3.59)$$

$$p_{ht} \in \{0, 1\} \quad \text{for } h = 1, \dots, q \text{ and } t = 1, \dots, \bar{T} \quad (3.60)$$

The objective function (3.52) aims at determining the smallest possible finishing time for the dummy end job n and, hence, the smallest project completion time. The restrictions (3.53) ensure that in each period t at most one compatible set S_h is put into progress. Each job j has to be processed for exactly d_j periods which is guaranteed by the duration constraints (3.54). The restrictions (3.55), (3.56), and (3.57) warrant that each job j is finished at the end of exactly one period and is executed without preemption. Finally, the precedence constraints are represented by the restrictions (3.58).

The number of variables and restrictions to be considered by the formulation (3.52)-(3.60) essentially depends on the number of compatible sets which can be determined for a problem instance. Like with the number of resource incompatible sets in formulation 5, the number of compatible sets is of the order $O(2^{n-2})$. Therefore, the applicability of this formulation is restricted to those problem instances where the actual number of such sets is low. However, as stated above, this formulation may be used to derive efficient arguments for computing lower bounds on the minimal project completion time (cf. Section 4.1.2.3).

Remark 3.5. The formulations 1 to 4 have been tested by the author concerning their efficiency using the general purpose optimization software CPLEX. Preliminary computational results reveal the formulations 1 and 3 to be most efficient. Formulation 2 clearly performs the worst. The other two formulations are relevant only in context of bound computation due to their size and complexity.

3.2.2 The Generalized Resource-Constrained Project Scheduling Problem (GRCPSP)

Though the RCPSP has attracted a large amount of research, its assumptions are rather restrictive. First of all, it only considers precedence relationships of

the simple finish-to-start type. However, as already indicated in Section 2.1.2, in many practical settings this type of precedence relationship is not sufficient to represent the corresponding real-world situations properly. For this reason, the precedence diagramming method has been developed which, e.g., permits (partially) executing a job in parallel to a predecessor (cf. Section 2.1.2, pp. 37). Second, RCPSP assumes constant availabilities of resources during the planning horizon. In practice, these availabilities are usually subject to predictable fluctuations, e.g., due to planned maintenance of equipment, vacations, or participation of workers and machines in other projects. Finally, release and due dates for single jobs can not be considered within RCPSP which represent important real-world restrictions. This restrictiveness is overcome by the generalized resource-constrained project scheduling problem (GRCPSP) which can be described as follows (cf. Section 4.3, pp. 149, for an example):

During a planning horizon \bar{T} with periods $t = 1, \dots, \bar{T}$, a single project has to be realized. Like with RCPSP, a set of jobs $J = \{1, \dots, n\}$ with fixed durations d_j have to be performed to finish the project. Additionally, each job j must be executed during a prespecified *time window* $[rd_j, dd_j]$ which is restricted by a given release date rd_j and a given due date dd_j . That is, at least rd_j periods must pass away before job j can be started and period dd_j is the latest one in which job j can be finished. Again, preemption of jobs is not allowed, i.e., whenever a job has been started, its execution must not be interrupted before its completion. In contrast to RCPSP, the precedence constraints are given in form of a *start-to-start project network*.

Like with RCPSP, executing the jobs requires several of m renewable resource types. In each period a job j is processed, it uses a constant amount of u_{jr} units of resource type r . However, the resource types $r = 1, \dots, m$ are available with time-varying amounts of a_{rt} units in the periods $t = 1, \dots, \bar{T}$.

The greater modeling flexibility provided by GRCPSP may, e.g., be favorably used for solving production scheduling problems involving sequence-independent set-up times as well as transfer and process batches (cf. Demeulemeester and Herroelen (1996 b)). Sprecher (1994, chapter 2) shows how the simple assembly line balancing problem of type 1 (cf. Scholl (1999, section 2.2.)) can be modeled as a GRCPSP. An application in the field of scheduling medical research experiments is described in Hartmann (1997 b). Furthermore, the high practical relevance of GRCPSP is confirmed by the fact that it is commonly

used as a basic optimization model within most commercial project scheduling software (e.g., MS Project, CA Superproject, Project Scheduler 7).

3.2.2.1 Properties of GRCPSP

Within the precedence diagramming method, also start-to-finish, finish-to-start as well as finish-to-finish precedence relationships with minimum time lags λ_{ij}^{SF} , λ_{ij}^{FS} , and λ_{ij}^{FF} are considered. These minimum time lags can easily be transferred into start-to-start ones using the following formulae (cf. Bartusch et al. (1988), Neumann and Schwindt (1997), and Domschke and Drexel (1998, p. 93)):

$$\lambda_{ij} = \lambda_{ij}^{SF} - d_j; \quad \lambda_{ij} = \lambda_{ij}^{FS} + d_i; \quad \lambda_{ij} = \lambda_{ij}^{FF} + d_i - d_j \quad (3.61)$$

Restricting to $\lambda_{ij} \geq 0$ does not impose a real restriction in project scheduling practice. The terms "predecessor" and "successors" already imply that the execution of a successor can not begin before all its predecessors have been at least started (cf. Demeulemeester and Herroelen (1997 a)). The same is true for the restriction to a non-cyclical project network. These assessments are underlined by the fact, that all major commercial software packages for project management share these restrictions.

Like with RCPSP, the number of variables which have to be defined within mathematical formulations for GRCPSP can considerably be reduced by computing time windows for jobs by a forward and a backward pass. However, the corresponding formulae (3.4) and (3.5) have to be modified in order to cope with start-to-start minimum time lags as well as release and due dates, respectively. Starting with $ES_1 = EF_1 = 0$, earliest starting and finishing times are computed as follows:

$$ES_j = \max\{rd_j, \max\{ES_i + \lambda_{ij} \mid i \in P_j\}\} \quad \text{for } j = 2, \dots, n \quad (3.62)$$

$$EF_j = ES_j + d_j \quad \text{for } j = 2, \dots, n \quad (3.63)$$

Note that by incorporating the release dates rd_j in the computation of the earliest starting times, introducing corresponding restrictions in the mathematical formulations can be avoided. The same is true for the due dates dd_j when they are considered within the backward pass which is started with $LF_n = LS_n = \bar{T}$:

$$LS_j = \min\{dd_j - d_j, \min\{(LS_h - \lambda_{jh}) \mid h \in F_j\}\} \text{ for } j = n-1, \dots, 1 \quad (3.64)$$

$$LF_j = LS_j + d_j \quad \text{for } j = n-1, \dots, 1 \quad (3.65)$$

3.2.2.2 Formulations

We renounce giving explicit mathematical formulations for GRCPSP, because the formulations 1 to 4 given for RCPSP can easily be adapted to this problem. For this purpose, only the precedence and the resource constraints have to be modified, respectively. The formulae (3.66) and (3.67) exemplarily show how this can be done for the formulation 1 ((3.8)-(3.12)). Between the scheduled starting time of a job j and the scheduled starting time of one of its predecessors i a minimum time lag of λ_{ij} periods has to be observed. The resource constraints have to refer to the current availability a_{rt} of a resource type r in a period t instead of the constant one a_r .

$$\sum_{t=EF_j}^{LF_j} (t-d_j) \cdot x_{jt} - \sum_{t=EF_i}^{LF_i} (t-d_i) \cdot x_{it} \geq \lambda_{ij} \quad \text{for } j \in J \text{ and } i \in P_j \quad (3.66)$$

$$\sum_{j \in E(t)} u_{jr} \cdot \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_j-1, LF_j\}} x_{jq} \leq a_{rt} \quad \text{for } r \in R \text{ and } t \in [1, \dots, \bar{T}] \quad (3.67)$$

In contrast to the formulations 1 to 4, adapting the formulations 5 and 6 is not worthwhile, because they are based on determining all resource incompatible and compatible sets in advance. Since this has to be done for each period separately due to the fluctuations in the resource availability, this would further reduce their applicability dramatically.

3.2.3 Problem Complexity

When dealing with optimization problems, their complexity in terms of computation time and storage space required for solving them with a computer program, i.e., a particular algorithm, is of major interest. In this context, the *complexity theory* which is based on a mathematical framework developed by logicians and computer scientists to study the complexity of algorithmic problems has proven very useful (cf., e.g., Cook (1971) for some seminal work).

This section presents a short overview of this theory and relates it to resource-constrained project scheduling. For excellent introductions and reviews, we refer to Garey and Johnson (1979), Papadimitriou and Steiglitz (1982, chapter 15) and Papadimitriou (1994). Note that in general, the computation time requirements are deemed to be the (main) limiting factor deciding whether an algorithm is suited for solving a problem or not.

In complexity theory, a strict distinction between the terms problem and instance is made. The term "problem" (also called *problem type*) refers to a generic description of the problem itself. The term "instance" is used for an actual problem for which the values of all parameters (input data such as the job durations) have been defined. With each problem instance, there is an associated size which corresponds to the length of the data string required to specify the input data. Of course, this length strongly depends on the encoding used. However, in practice, the size of a problem instance is usually characterized by the number n of parameters, assuming that these are represented by a fixed number of digits. This is sufficiently accurate for making distinctions between the complexities of different problems. In the context of scheduling, the problem complexity commonly refers to the number n of jobs and the number m of resource types to be considered.

When comparing the time requirements of different algorithms for solving a certain problem type, this is most commonly done by determining an upper bound on the maximum number of computational steps which have to be performed in order to determine an (optimal) solution for a respective instance. This, in turn, requires a definition of a computational step. Most commonly, operations like assigning values to variables, basic algebraic functions like multiplying two values, and comparisons of values are considered to be such elementary operations. Based on such computational steps, the worst case upper bound is described by defining a *time complexity function*. Usually, only the order of this function is of interest. A function $f(n)$ is said to be of the *order* $O(g(n))$, if $c \cdot f(n) \leq g(n)$ with c being a sufficiently large positive constant.

In general, two basic classes of algorithms can be distinguished concerning time complexity functions. Within *polynomial time algorithms*, the number of computational steps is limited by a polynomial function which depends on the problem size, i.e., the length n of the input data. That is, the time complexity function is of the order $O(n^q)$ with q denoting a positive constant. By the way of contrast, the number of computational steps exponentially increases with the

problem size in *exponential time algorithms*. For example, the time complexity function of such an algorithm may be of the order $O(q^n)$.

Basically, problem types may be subdivided into *optimization* and *decision* problems. The latter class of problems has only two possible solutions and requires either a "yes" or a "no" answer. In the case of a minimization problem, a corresponding decision problem can be derived by limiting the maximum objective function value to a particular constant. This is achieved by omitting the objective function and introducing a corresponding constraint. For example, a decision problem arising from RCPSP could consist in answering the question, whether a schedule observing the precedence and the resource constraints exists for a particular project completion time or not.

The theory of NP-completeness is based on distinguishing the following classes of decision problems:

Definition 3.7. The *problem class NP* covers all decision problems for which there exists a polynomial time algorithm verifying whether the answer is correct or not for a given "yes" solution. This class can be further subdivided as follows:

The *class P* is composed of all those problems within NP a solution, i.e., a correct answer, of which can be determined by a polynomial time algorithm. By the way of contrast, the *class NP-complete (NPC)* consists of the problems contained in NP for which no polynomial algorithm solving all instances is known. All problems of the latter class can be transformed into each other in polynomial time such that after finding a polynomial time algorithm for one of these problems, the remaining ones can be solved in polynomial time, too.

Note that one of the most important issues open in combinatorial optimization is the question whether or not $P = NP$. If P was equal to NP , then P and NPC would not be disjoint. In general, the class NPC represents the "hardest" problems contained within NP .

A fundamental concept in complexity theory is reduction. Very often, it occurs that one problem type is a special case of another type or vice versa. For example, the job shop scheduling problem represents a special case of RCPSP. A transformation of a problem Π into a problem Π' is termed *reduction* if Π' is a special case of Π . This concept is often used to prove that a decision problem belongs to NPC by demonstrating that it is in NP and can be reduced to another NP -complete problem in polynomial time.

Though the above argumentation refers to decision problems only, the results can also be applied to other problems not part of NP including optimization problems.

Definition 3.8. The problem class *NP-hard* consists of those problems (not necessarily contained within NP) for which no polynomial time algorithm determining an optimal solution has been developed yet. The problems for which at least a feasible solution can be determined in polynomial time are called *NP-hard* in the ordinary sense or simply *NP-hard*. Those where even this is impossible are termed *NP-hard in the strong sense*. Note that any *NP-hard* problem can be transformed in polynomial time into a decision problem of *NPC*.

The RCPSP can be reduced to the well-known job shop scheduling problem (cf., e.g., Schrage (1970), Baker (1974), Drexel (1990), Sprecher (1994, chapter 2)). Since this problem is *NP-hard*, RCPSP is *NP-hard*, too (cf. Garey et al. (1976) and Blazewicz et al. (1983)). However, it is not *NP-hard* in the strong sense, because a feasible solution can be obtained by computing a topological order of the jobs in polynomial time and scheduling the jobs in this order one after each other. By the way of contrast, this is true for GRCPSP. This can be shown by constructing a GRCPSP instance as follows. An RCPSP instance is solved to optimality. Subsequently, the finish-to-start project network is transformed into a start-to-start one and a due date is introduced for the dummy end job corresponding to the optimal project completion time determined for the RCPSP instance. Obviously, for this GRCPSP instance, a precedence and resource feasible schedule can not be obtained in polynomial time, because it requires solving the *NP-hard* RCPSP. For a more detailed discussion on complexity results for RCPSP, we refer to Schirmer (1996 a).

3.3 Extensions of the Basic Models

In the following, further possible extensions of RCPSP which have been considered in the literature are briefly introduced. Furthermore, references on literature describing appropriate solution approaches are given. Except for the multi-mode version, we renounce developing complete mathematical formulations but restrict to show how the ones presented for RCPSP can be adapted accordingly.

3.3.1 Preemption

In practice, the assumption that a job which has been started once must not be interrupted before its termination often is not justified. For example in a software project, a programmer may stop with implementing parts of the program to be developed in order to participate in an audit and continue later on. Therefore, it can be useful to extend RCPSP by allowing preemption of jobs.

Basically, this can be achieved by splitting each job j with a duration d_j into d_j different sub-jobs $j_1, \dots, j_h, \dots, j_{d_j}$ with durations of a single period, respectively, and introducing finish-to-start precedence relationships between all pairs of sub-jobs (j_h, j_{h+1}). Furthermore, all predecessors of job j contained in P_j are defined to be the predecessors of the sub-job j_1 and all followers contained in F_j become followers of the sub-job j_{d_j} . If preemption of a job is only possible after a certain, prespecified number of periods, this can be modeled by choosing the durations of the sub-jobs accordingly. However, this technique has the disadvantage that the number of jobs is considerably increased in the resulting finish-to-start project network.

Therefore, it may be appropriate to adapt mathematical formulations as well as solution procedures for RCPSP such that they can immediately cope with pre-emption. For formulation 2 ((3.14)-(3.19), p. 81), this is achieved by dropping the restrictions (3.15). Exact procedures tailored for solving RCPSP with pre-emption have been proposed by Kaplan (1988) and Demeulemeester and Herroelen (1996 a). Note that RCPSP with preemption is NP-hard, too. A special version with $a_r = 1$ for $r = 1, \dots, m$ is considered by Bianco et al. (1999).

3.3.2 Multiple Modes

In many real-world situations, the duration of a job can be decreased at the expense of providing additional resources. For example, an excavation may be digged by five construction workers within a week, whereas the same work may be performed by an excavator within a single day. This relationship between resource usage and the duration of a job can be modeled using *modes* resulting in the multi-mode RCPSP which can described as follows.

Completing a project requires the execution of a set J of jobs, the precedence relationships between which are represented in form of a finish-to-start project

network. Each job $j \in J$ can be processed in one of several different modes $M_j = \{1, \dots, p_j\}$. Associated with each mode $s = 1, \dots, p_j$ is a duration d_{js} . In contrast to the single-mode case, i.e., RCPSP, the execution of a job may require renewable and non-renewable resource types as well (cf. Section 1.5.3, pp. 24). The latter class of resource types (e.g., materials, budget) is not considered in the single-mode case because either their total availability is sufficient for terminating the project or not. The index sets of the renewable and non-renewable resource types are denoted by R and N , respectively. The per period availability of a renewable resource type $r \in R$ is expressed by a_r . The same notation is used for the total availability the non-renewable resource types with $r \in N$. The per period demands of renewable resource types and the total resource usages of non-renewable resource types which are caused by a job j performed in a mode s are given by u_{jsr} for $r \in R$ and $r \in N$, respectively.

The multi-mode RCPSP is to assign a starting time as well as an execution mode to each job $j \in J$ such that the precedence and resource constraints are not violated and the project completion time is minimized. Note that doubly-constrained resource types can be modeled by introducing a corresponding renewable as well as a non-renewable resource type.

In the following, we adapt formulation 1 ((3.8)-(3.12), pp. 79) to the multi-mode RCPSP. For this purpose, time windows for the execution of the jobs have to be computed by a modified forward and backward pass. A simple upper bound on the project completion time can be obtained by summing up the longest possible durations of all jobs according to (3.3), p. 78. Subsequently, earliest and latest starting and finishing times can be computed by assuming that each job is executed in a mode with the shortest duration using the formulae (3.4) and (3.5). Furthermore, we extend the definition of the 0-1 variables x_{jst} as follows:

$$x_{jst} = \begin{cases} 1, & \text{if job } j \text{ is finished in mode } s \text{ at the end of period } t \\ 0, & \text{otherwise} \end{cases} \quad \text{for } j \in J, s \in M_j, \text{ and } t \in [EF_j, LF_j] \quad (3.68)$$

As a matter of convenience, the dummy start and end job can only be processed in a single mode $s = 1$ with a duration of $d_1 = d_n = 0$, respectively. Then, using the above parameters and variables, the multi-mode RCPSP can be formulated by the following binary model:

$$\text{Minimize } CT(x) = \sum_{\substack{t=EF_n \\ t=EF_n}}^{LF_n} t \cdot x_{n1t} \quad (3.69)$$

subject to

$$\sum_{\substack{s=1 \\ s=1 \\ t=EF_j}}^{p_j} \sum_{t=EF_j}^{LF_j} x_{jst} = 1 \quad \text{for } j \in J \quad (3.70)$$

$$\sum_{\substack{s=1 \\ s=1 \\ t=EF_i}}^{p_j} \sum_{t=EF_i}^{LF_i} x_{ist} \leq \sum_{\substack{s=1 \\ s=1 \\ t=EF_j}}^{p_j} \sum_{t=EF_j}^{LF_j} (t - d_{js}) \cdot x_{jst} \quad \text{for } j \in J \text{ and } i \in P_j \quad (3.71)$$

$$\sum_{j \in E(t)} \sum_{s=1}^{p_j} u_{jsr} \cdot \sum_{\substack{q=\max\{t, EF_j\} \\ q=q}}^{\min\{t+d_{js}-1, LF_j\}} x_{jsq} \leq a_r \quad \text{for } r \in R \text{ and } t \in [1, \dots, \bar{T}] \quad (3.72)$$

$$\sum_{j=1}^n \sum_{s=1}^{p_j} u_{jsr} \cdot \sum_{\substack{q=EF_j \\ q=q}}^{LF_j} x_{jsq} \leq a_r \quad \text{for } r \in N \quad (3.73)$$

$$x_{jst} \in \{0, 1\} \quad \text{for } j \in J, s \in [1, \dots, p_j], \text{ and} \\ t \in [EF_j, \dots, LF_j] \quad (3.74)$$

The objective function (3.69) minimizes the finishing time of the dummy-end job n. Each job has to be assigned exactly one mode and one finishing time. This is guaranteed by the assignment constraints (3.70) together with the restrictions (3.74). In order to observe the precedence constraints, no job must be begun before all its predecessors have been completed which is ensured by the restrictions (3.71). Finally, resource constraints have to be observed for both, renewable and non-renewable resource types (restrictions (3.72) and (3.73)). Note that the multi-mode version of RCPSP is NP-hard in the strong sense (cf. Kolisch (1995, section 2.3.2), Schirmer (1996 b)).

Methods for computing lower bounds on the smallest possible project completion time are described in Vercellis (1994) and Maniezzo and Mingozi (1999). Exact methods for solving the multi-mode RCPSP have, e.g., been presented by Talbot (1982), Patterson et al. (1989), Patterson et al. (1990), Speranza and

Vercellis (1993), Sprecher (1994, chapter 5), Hartmann and Sprecher (1996), Sprecher et al. (1997), Sprecher and Drexel (1998), and Pesch (1999). Hartmann and Drexel (1998) review the different approaches and provide comprehensive computational results comparing their efficiency. For priority-rule based multi-pass heuristics we refer to Drexel (1991), Drexel and Grünwald (1993), Boctor (1993), Kolisch (1995, chapter 6) as well as Boctor (1996 a). Kolisch and Drexel (1997) present a local search based algorithm. An approach based on a local constraint based analysis is described in Özdamar and Ulusoy (1994). Boctor (1996 b) develops a simulated annealing approach. Genetic algorithms are proposed by Mori and Tseng (1997) and Hartmann (1997 a), with the latter one outperforming all heuristic approaches presented so far. Further generalizations of the basic multi-mode RCPSP are, e.g., considered in Ahn and Selcuk Eren-guc (1998) and Bianco et al. (1998).

3.3.3 Maximum Time Lags

GRCPSP extends RCPSP by allowing for several types of precedence relationships associated with minimum time lags as defined by the precedence diagramming method. Furthermore, as already described at the end of Section 2.1.2, pp. 37, also start-to-start precedence relationships with maximum time lags $\bar{\lambda}_{ij}$ limiting the maximum number of periods which can pass between the start of a job i and the start of its successor j may be considered. Such maximum time lags have been introduced for the first time in the context of the meta-potential method (Roy (1964)). Possible applications, e.g., in make-to-order production, are presented in Neumann and Schwindt (1997).

In order to integrate maximum time lags $\bar{\lambda}_{ij}$ into the formulations 1 to 4, additional precedence relationships have to be defined as done for formulation 1 ((3.8)-(3.12), pp. 79) below. \bar{P}_j denotes the predecessors of a job j for which maximum time lags to job j have to be observed.

$$\sum_{\substack{LF_j \\ t=EF_j}} (t-d_j) \cdot x_{jt} - \sum_{\substack{LF_i \\ t=EF_i}} (t-d_i) \cdot x_{it} \leq \bar{\lambda}_{ij} \quad \text{for } j \in J \text{ and } i \in \bar{P}_j \quad (3.75)$$

A more convenient way consists in transforming each maximum time lag $\bar{\lambda}_{ij}$ between two jobs i and j into a minimum one. This can be achieved by defining a precedence relationship in the reverse direction (j, i) and assigning a minimum time lag of $\lambda_{ji} = -\bar{\lambda}_{ij}$. The correctness of this transformation can easily be

verified. With SS_i and SS_j denoting the scheduled starting time of the jobs i and j , respectively, the restrictions (3.75) may be rewritten as $SS_j - SS_i \leq \bar{\lambda}_{ij}$ for $j \in J$ and $i \in P_j$. Multiplying this term by -1 yields $SS_i - SS_j \geq -\bar{\lambda}_{ij}$. However, as a consequence, the corresponding start-to-start project network is no longer non-cyclical. Therefore, solution methods designed for solving RCPSP and GRCPSP may not be applied, because they usually exploit that no successor j of a job i may start before the job itself. This led to the development of a number of specialized procedures.

Procedures for computing lower bounds on the minimum project completion time for RCPSP with maximum time lags are discussed by Heilmann and Schwindt (1997) as well as Schwindt (1998 c). They apply the destructive improvement method which has been introduced by Klein and Scholl (1999) for RCPSP and which is comprehensively discussed in Section 4.2. Branch and bound procedures are presented by Bartusch et al. (1988), De Reyck (1998), De Reyck and Herroelen (1998), Dorndorf et al. (1998), Möhring et al. (1998), and Schwindt (1998 a, c). Different priority-rule based heuristics can be found in Zhan (1994), Neumann and Zhan (1995), Brinkmann and Neumann (1996), and Franck and Neumann (1997) with the last one containing the most recent results. Meta-heuristic based procedures are discussed in Franck and Selle (1998). A truncated branch and bound procedure is introduced by Schwindt (1998). A survey on the different approaches as well as computational evaluations examining their efficiency are contained in Neumann and Zimmermann (1999). Heilmann (1998) describes a branch and bound procedure which additionally considers the possibility of performing jobs in multiple modes as described in Section 3.3.2.

3.3.4 State Preserving Jobs

A state preserving job has a variable duration which depends on the scheduled finishing and starting times of its predecessors and successors, respectively. That is, it starts as soon as the execution of all its predecessors has been terminated and finishes immediately before the first of its successors is begun. State preserving jobs provide a powerful modeling tool, if resources cannot be released between the execution of certain jobs. The following example represents a possible application for the use of state preserving jobs.

For assembling a make-to-order product, it has to be processed on two different machines successively, which is represented by a job i and a successor job j in the project network. After leaving the first machine, the product possibly has to be stored, because its processing can not be continued immediately due to the second machine still being occupied by other products. If the available storage space is restricted, this has to be modeled by a respective resource constraint. Furthermore, an additional job which accordingly uses this resource type has to be introduced between the jobs i and j . Since the duration of this job depends on the finishing time of job i and the starting time of job j , its duration can not be determined in advance such that a state preserving job is required.

In order to include state preserving jobs in the formulations 1 to 4, only the resource constraints have to be modified which is exemplarily shown for formulation 1 ((3.8)-(3.12), pp. 79). To ease the presentation, we restrict ourselves to the case, that each state preserving job has exactly one predecessor and one successor. Additionally, the following notation has to be introduced. Each state preserving job is represented by a pair of jobs $(i,j) \in A$. During each period of its "processing", it requires \bar{u}_{ijr} units of each resource type $r = 1, \dots, m$. The set of all state preserving jobs is denoted by SP . With these parameters, the restrictions (3.11) can be modified as follows:

$$\left. \begin{aligned} & \sum_{j \in E(t)} u_{jr} \cdot \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_j-1, LF_j\}} x_{jq} + \\ & \sum_{(i,j) \in SP} \bar{u}_{ijr} \cdot \left(\sum_{q=EF_i}^{\min\{t-1, LF_i\}} x_{iq} - \sum_{q=EF_j}^{\min\{t+d_j-1, LF_j\}} x_{jq} \right) \leq a_r \end{aligned} \right\} \begin{array}{l} \text{for } r \in R \text{ and} \\ t \in [1, \dots, \bar{T}] \end{array} \quad (3.76)$$

The resulting restrictions (3.76) may be explained as follows. For a resource type $r \in R$, the first sum represents the total demand of the standard jobs executed in period t . The second sum determines the resource usage of the state preserving jobs. A state preserving job (i,j) is active in period t , if job i has been finished until the end of period $t-1$ the latest and job j has not been started at the beginning of period t or earlier.

Several computer software packages offer the possibility to define state preserving jobs though the names used may differ (cf., e.g., Project Scheduler by

Scitor). However, to the best of our knowledge, no scientific literature exists addressing RCPSP with state preserving jobs.

3.3.5 Further Extensions

In this section, we give hints on further extensions of RCPSP which are not covered in detail within this book. This is due to the fact that neither these extensions have attracted a great interest in the literature so far nor they have been integrated into commercial project management software. Priority-rule based heuristics for solving the RCPSP when set-up times have to be considered for a subset of the provided resources are discussed in Kolisch (1995, chapter 8). Shewchuck and Chang (1995) propose the use of the recyclable resource types. These are resource types for which completely consumed units can be processed (recycled) in some manner in order to obtain new ones. Böttcher et al. (1999) introduce partially renewable resource types. Their basic idea consists of limiting the availability of renewable resource types not for single periods but for subsets of periods. Both, priority-rule based heuristics as well as a branch and bound procedure are presented. Possible applications of partially renewable resources, e.g., in the field of shift scheduling, are described in Schirmer and Drexel (1997) and Drexel et al. (1998 a). For multi-mode RCPSP, Salewski et al. (1995, 1997) describe a heuristic solution method for the case that subsets of jobs have to be performed in one and the same mode. Such mode-identity constraints may be favorably used, e.g., in order to model problems arising in the context of audit-staff scheduling (cf. Drexel et al. (1998 a)). Yoo et al. (1995) describe an exchange heuristic for a modified multi-mode problem. Finally, doubly constrained, continuously divisible resource types, such as energy, are considered in Weglarz (1981) and Weglarz (1989).

3.4 Related Project Scheduling Problems

In this section, mathematical optimization problems are described which arise if the constraints of RCPSP are combined with another objective function than the minimization of the project completion time. For this purpose, mathematical formulations are developed based on formulation 1 for RCPSP. Furthermore, references on according literature are given.

3.4.1 The Time-Constrained Project Scheduling Problem

Besides resource-constrained project scheduling, time-constrained project scheduling represents another important problem type in practical project planning (cf. Section 2.3.3). Sometimes, a given deadline \bar{T} on the project completion time of a project may be too tight in order to be realized with the resources initially provided. In this case, additional resources have temporarily to be allocated in certain periods resulting in additional cost depending on the respective resource type. The time-constrained project scheduling problem (TCPSP) consists of determining a schedule such that the project is completed on time and the total additional cost are minimized. In the following, we describe how to adapt formulation 1 to the TCPSP.

First of all, the resource constraints of formulation 1 have to be modified in order to model the possibility of extending the resource availabilities in certain periods. Assuming the additional units of resource type r provided in period t to be denoted by w_{rt} , the restrictions (3.11) have to be replaced by (3.77):

$$\left(\sum_{j \in E(t)} u_{jr} \cdot \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_j-1, LF_j\}} x_{jq} \right) - w_{rt} \leq a_r \quad \text{for } r = 1, \dots, m \text{ and } t \in [1, \dots, \bar{T}] \quad (3.77)$$

In practice, the maximum amount by which the availability of a resource type may be extended is usually limited. For example, overtime may be restricted to at most 20% of the standard working time. With w_r^{\max} denoting the maximal amount of extension possible for a resource type r , we yield the following additional restrictions:

$$0 \leq w_{rt} \leq w_r^{\max} \quad \text{for } r = 1, \dots, m \text{ and } t \in [1, \dots, \bar{T}] \quad (3.78)$$

Note that no explicit restriction is required to limit the maximum project completion time, if the time windows are calculated properly (cf. Remark 3.1, p. 78). In order to define the objective function, the cost c_r for providing an additional unit of resource type r have to be specified. Then, the time-constrained project scheduling problem may be formulated as follows:

$$\text{Minimize } O(\mathbf{x}, \mathbf{w}) = \sum_{r=1}^m o_r \cdot \sum_{t=1}^{\bar{T}} w_{rt} \quad (3.79)$$

subject to (3.9), (3.10), (3.77), (3.78), and (3.12)

Though TCPSP is of high practical relevance, it has been considered in the literature only rarely. A mixed-integer formulation for this NP-hard problem corresponding to the one above is given by Deckro and Hebert (1989). A heuristic procedure is proposed in Kolisch (1995, chapter 7). Furthermore, the priority-rule based heuristics initially proposed for resource leveling by Neumann and Zimmermann (1997, 1998, 1999) and Zimmermann (1997) may also be applied, even considering minimum and maximum time lags (cf. Section 3.4.2). The same is true for the branch and bound procedure developed by Zimmermann and Engelhardt (1998).

Within commercial project management software, the parameter values of TCPSP, e.g., overtime cost, can usually be specified. However, the capabilities of software packages concerning time-constrained scheduling are very restricted. The only support they offer consists of automatically calculating the total cost for a given schedule.

3.4.2 The Resource Leveling Problem

The resource leveling problem (RLP) is closely related to TCPSP. Basically observing the same set of constraints, in particular a given deadline on the project completion time, the major difference concerns the objective functions used. Within resource leveling, the objective is to determine a schedule such that fluctuations in the period usages of resources are minimized. This follows the idea that a more steady usage rate leads to lower cost (cf. Section 1.5.3). Therefore, after determining the shortest possible project completion time CT for a problem instance by solving RCPSP, RLP may be considered subsequently with $\bar{T} := CT$ as deadline.

In order to formulate according objective functions mathematically, the period demand of each resource type $r \in R$ has to be determined which is possible by adapting the resource constraints. For formulation 1, the restrictions (3.11) have to be modified as follows containing a_{rt} as a variable instead of as a fixed parameter:

$$\sum_{j \in E(t)} u_{jr} \cdot \sum_{q=t}^{t+d_j-1} x_{jq} - a_{rt} = 0 \quad \text{for } r \in R \text{ and } t \in [1, \dots, \bar{T}] \quad (3.80)$$

Now, the most commonly considered resource leveling problem can be defined as follows, where w_r is a weight connected with resource type r :

$$\text{Minimize } L(\mathbf{x}, \mathbf{a}) = \sum_{r=1}^m w_r \cdot \sum_{t=1}^{\bar{T}} a_{rt}^2 \quad (3.81)$$

subject to (3.9), (3.10), (3.80), and (3.12)

Further possible objective functions as well as their properties are, e.g., discussed in Neumann and Zimmermann (1997). Different priority-rule based procedures have been proposed by Burgess and Killebrew (1962), Levy et al. (1962), Moodie and Mandeville (1966), Woodworth and Willie (1975), Harris (1990) as well as Takamoto et al. (1995). Savin et al. (1996) report on experiences with a neural network approach. Different exact approaches are presented in Ahuja (1976), Easa (1989), Bandelloni et al. (1994) and Younis and Saad (1996). Neumann and Brinkmann (1996), Neumann and Zimmerman (1997, 1998, 1999), and Zimmerman (1997) propose priority-rule based heuristics also considering minimum and maximum time lags between jobs. A tabu search procedure is described in Neumann and Zimmerman (1999). Finally, Zimmerman and Engelhardt (1998) have developed an exact procedure for this extension.

3.4.3 The Resource Investment Problem

The resource investment problem (RIP) is obtained from RCPSP by introducing another objective function. Like TCPSP and RLP, it addresses the case that a deadline \bar{T} on the project completion time is given. However, it usually does not arise during the project planning phase but already in the definition one. Within the budgeting process, it has to be decided in which amount the resources required for the execution of a project will be provided, e.g., how many machines will be purchased or how many workers will be hired. For this purpose, usually estimates c_r are specified which describe the expected cost for supplying a single unit of each resource type r for all periods of the planning horizon

$1, \dots, \bar{T}$. Furthermore, in order to express the long term effect of these decisions, it is assumed that the initially supplied availability a_r of a resource type r can not be changed during the execution of the project. Then RIP consists of determining a schedule as well as the amount to be provided of each resource type such that the project can be completed on time at minimum total cost without violating resource or precedence constraints.

Based on the above parameters, RIP can be defined as given below using the decision variables and restrictions of formulation 1. Note that the availabilities a_r of the resource types $r \in R$ become additional variables. The deadline on the project completion time is considered within the calculation of the time windows (cf. Remark 3.1).

$$\text{Minimize } C(x) = \sum_{r=1}^m c_r \cdot a_r \quad (3.82)$$

subject to (3.9)-(3.12)

Branch and bound procedures for this NP-hard problem have been presented by Möhring (1984) and Demeulemeester (1995). Drexl and Kimms (1998 b) describe the computation of lower bounds using Lagrangean relaxation. Finally, Nübel (1998) presents a branch and bound procedure which can also cope with maximum time lags.

3.4.4 The Net Present Value Problem

In long term projects, significant levels of cashflows may incur. Most often, these cash flows are somehow related to the execution of jobs. For example in case of performing a project on request, progress payments by the contractor may be tied to the completion of a subset of jobs. In turn, the execution of some jobs may not be possible until certain equipment or materials have been purchased. Therefore, when defining due dates for project milestones in the definition phase, an appropriate objective function may consist of finding a schedule such that the net present value of the project is maximized (cf. Section 1.3.2, pp. 8). Seminal work on this problem has been performed by Russell (1970) and Grinold (1972) (cf. Section 1.4.4, pp. 19, for a survey on corresponding literature). However, they restrict to considering the case where resource availabilities are not limited (NPVP). Though in contrast to RCSPS the planning horizon addressed is considerably longer, this assumption is not always justified for

all resource types required for completing the project successfully. Then, the resource-constrained net present value problem (RCNPVP) arises.

In order to state RCNPVP more formally, we define cf_j to be the cashflow incurring at the end of executing a job j . Furthermore, let ir denote the interest rate by which the cashflows are discounted. Note that cashflows arising during the execution of a job can easily be aggregated to a final one. Then, using the restrictions of the RCPSP formulation 1, RCNPVP can be formulated as follows:

$$\text{Maximize } NPV(x) = \sum_{j=1}^n \sum_{t=EF_j}^{LF_j} cf_j \cdot (1+ir)^{-t} \cdot x_{jt} \quad (3.83)$$

subject to (3.9)-(3.12)

In case of the project having a negative net present value independent of the schedule chosen, the project completion time will be increased to the upper bound $\bar{T} = LF_n$ in order to keep the loss as small as possible.

Priority-rule based heuristics are developed by Russell (1986), Smith-Daniels and Aquilano (1987), Padman and Smith-Daniels (1993), Padman et al. (1997), and Neumann and Zimmermann (1998, 1999). Heuristics for the RCNPV-problem with capital constraints are proposed by Smith-Daniels and Smith-Daniels (1987) as well as Smith-Daniels et al. (1996). A simulated annealing approach is presented by Yang et al. (1995), whereas Icmeli and Selcuk Erenguc (1994) describe a tabu search procedure. Exact algorithms using branch and bound are proposed by Yang et al. (1993 a), Icmeli and Selcuk Erenguc (1996), De Reyck and Herroelen (1998 b) and Baroum and Patterson (1999). Like with RCPSP, a possible extension of RCNPVP consists of allowing multiple execution modes for jobs (cf. Section 3.3.2). Corresponding priority-rule based heuristics and an improvement procedure can be found in Sung and Kim (1994) and in Ulusoy and Özdamar (1995), respectively.

A further possible extension considers penalty payments in case of jobs being delayed beyond a prespecified due date. A corresponding mathematical formulation is given in Doersch and Patterson (1977). Finally, priority-rule based heuristics taking into account periodical progress payments are presented in Sepil and Ortac (1997).

3.4.5 The Weighted Tardiness Problem

When scheduling is performed within a multi-project environment, it often occurs that due dates on the completion of sub-projects or of jobs have to be observed. If these due dates can not be met simultaneously due to the restricted availability of resources, a resource-constrained weighted tardiness problem (RCWTP) arises. Assuming that per period cost can be specified for delaying a job beyond its due date, e.g., due to penalty payments covenant upon with a customer, a schedule has to be determined such that the total cost are minimized without violating the resource and precedence constraints.

According to GRCPSP, the due date of a job j is defined by dd_j . The penalty cost incurring in each period by which the completion of the job is delayed beyond its due date are denoted by pc_j . Using these parameters, RCWTP may be formulated mathematically as follows:

$$\text{Maximize } PC(\mathbf{x}) = \sum_{j=1}^n \sum_{t=\max\{EF_j, dd_j\}}^{LF_j} pc_j \cdot (t - dd_j) \cdot x_{jt} \quad (3.84)$$

subject to (3.9)-(3.12)

Heuristic procedures for RCWTP can, e.g., be found in Patterson (1976), Kurtulus and Narula (1985), Weiss (1988), Kim and Schniederjans (1989), Morellin (1989), Mohanty and Siddiq (1989 a, 1989 b), Deckro et al. (1991), Kim and Leachman (1993), Lawrence and Morton (1993), and Leon and Balakrishnan (1995). Another important problem arising in this context consists of setting appropriate due dates in a multi-project environment where new projects may be started dynamically. According studies evaluating the efficiency of different due date setting rules can, e.g., be found in Dumond and Mabert (1988), Yau and Ritchie (1988), Bock and Patterson (1990), Dumond (1992), and Yang and Sum (1993). Applications of RCWTP are described by Tsubakitani and Deckro (1990) and Della Croce et al. (1993).

3.4.6 Further Resource-Constrained Project Scheduling Problems

Icmeli-Tukel and Rom (1997) present an approach which aims at increasing the quality of a project by minimizing the rework necessary. In Icmeli-Tukel and

Rom (1998), the authors indicate that this represents one of the most important objective functions in real-world project scheduling. Furthermore see Selcuk Erenguc and Icmeli-Tukel (1999).

In decision support systems for resource-constrained project scheduling, often multi-criteria objective functions are considered. According objective functions as well as decision support systems are, e.g., described in Slowinski (1981), Tavares (1986), Slowinski (1988), Davis et al. (1992), Nowicki and Smutnicki (1994), Rys et al. (1994), Serafini and Speranza (1994), Slowinski et al. (1994), Jüngen and Kowalczyk (1995), Norbis and Smith (1996), Özdamar and Ulusoy (1996 a), Nkasu and Leung (1997), Hapke et al. (1999), and Nabrzyski and Weglarz (1999).

Part II

Resource-Constrained Project Scheduling: Solution Methods

4 Lower Bound Methods

In this chapter, methods for computing lower bounds on the smallest possible project completion time of a project which has to be scheduled under resource constraints are described. The purpose behind developing such methods is two-fold. If heuristic procedures as described in Chapter 5 are applied for determining a feasible solution, the size of the gap between a computed lower bound value and the project completion time associated with the determined solution can be computed. This may provide a measure for judging whether investing additional planning effort in order to determine an improved solution is justified or not. In case of using exact procedures such as described in Chapter 6, according methods can be incorporated into the procedures in order to reduce the solution effort required.

The description is orientated on the discussion of two meta-strategies for computing lower bounds (for minimization problems). *Constructive (direct) methods* directly calculate a bound value by relaxing a problem and solving this relaxation. *Destructive improvement techniques*, which have been developed by the author, restrict a problem by setting a maximal objective function value F and try to contradict (destruct) the feasibility of this reduced problem. In case of success, F or even $F + 1$ is a valid lower bound value. In the Sections 4.1 and 4.2, the fundamental properties and differences of both meta-strategies are explained by applying them to RCPSP. For this problem, efficient lower bound arguments have been developed only recently. We describe extensions and new methods as well as techniques for reducing problem data which can be exploited within the destructive improvement framework. Though the destructive improvement technique is rather new, there are already a number of successful applications. These are briefly described at the beginning of Section 4.2. Finally, in Section 4.3, it is shown how these methods can be modified such that they can also be used for GRCPSP.

4.1 Constructive Lower Bound Methods for RCPSP

In this section, different constructive methods for computing lower bounds for RCPSP instances are presented. Section 4.1.1 reviews simple lower bound arguments which can be realized efficiently with small computational effort. In Section 4.1.2, more complex bound arguments are briefly discussed the application of which results in a much larger computational effort. For example, these approaches rely on linear programming and Lagrangean relaxation.

4.1.1 Simple Bound Arguments

In the following, different simple arguments for calculating lower bounds on RCPSP instances are introduced. Since most of these bound arguments are developed by utilizing relationships to other optimization problems, the description is organized along these problems.

In order to ease the presentation, we give an example for the application of each bound argument. Except for the bin packing bounds (cf. Section 4.1.1.2), this is done for the example instance already introduced in the Sections 2.1.2 and 2.3.1. For the sake of convenience, the corresponding finish-to-start project network is given once again in Figure 4.1. It consists of $n = 12$ jobs and considers one renewable resource type ($m = 1$) which is available with $a_1 = 4$ units per period. The resource usage u_{j1} of each job j is denoted below the corresponding node. Job 1 and job n are dummy start and end nodes with duration 0. An optimal solution for the corresponding problem instance is shown in Figure 2.7, p. 53.

4.1.1.1 Critical Path and Capacity Bounds

The following three lower bound arguments exploit the special problem structure of RCPSP.

Critical Path Bound (LBC1). The most obvious and most frequently used lower bound argument is based on omitting the capacity restrictions. The *minimal project duration* of this relaxed problem is obtained by computing the length of a *critical (longest) path* in the project network. As already described in the Sections 2.2.1 and 3.2.1.1, this can be done by performing a *forward pass* which can be realized in $O(n^2)$ time. This yields earliest starting and finishing times for all jobs with LBC1 (length of the critical path) being equal to the ear-

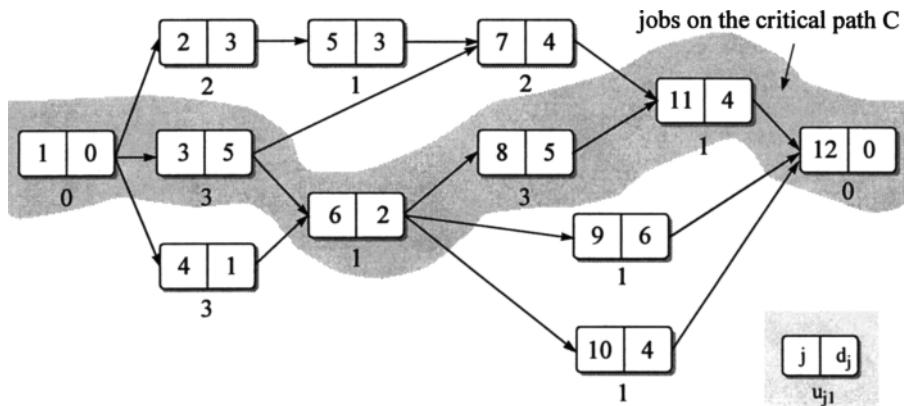


Figure 4.1. Critical path in the example project

liest starting time of the terminal dummy job. Subsequently, latest finishing and starting times may be computed by a *backward pass* beginning with $LF_n = LS_n = LBC1$. For each job j , the interval $[ES_j, LF_j]$ defines a *time window* in which j has to be processed if the project should be completed after $LBC1$ time units.

In our example, the critical path is $CP = \langle 1, 3, 6, 8, 11, 12 \rangle$ with a total duration of $LBC1 = 16$ periods (cf. Figure 4.1). The resulting time windows are given in Table 4.1 (the values e_j^{\max} are explained later).

Table 4.1. Time windows

j	1	2	3	4	5	6	7	8	9	10	11	12
d_j	0	3	5	1	3	2	4	5	6	4	4	0
ES_j	0	0	0	0	3	5	6	7	7	7	12	16
LF_j	0	5	5	5	8	7	12	12	16	16	16	16
e_j^{\max}	—	0	—	0	5	—	1	—	9	9	—	—

Capacity Bound (LBC2). This rather simple bound argument, which can be calculated in $O(m \cdot n)$ time, discards all but one resource type. A bound value is computed by (4.1) as the total requirement of this resource type divided by its per period availability and rounded up to the next larger integer. In the example, we obtain $LBC2 = \lceil 66/4 \rceil = 17$.

$$LBC2 = \max \left\{ \left[\left(\sum_{j=1}^n u_{jr} \cdot d_j \right) / a_r \right] \mid r = 1, \dots, m \right\} \quad (4.1)$$

Critical Sequence Bound (LBC3). This bound adds capacity considerations to the critical path bound (cf. Stinson et al. (1978)). Imagine the jobs of a critical path CP being scheduled one immediately after its predecessor. Processing some job $j \in J - CP$ in parallel to the critical path requires that within its time window $[ES_j, LF_j]$ there is an interval with minimum length d_j during which the residual capacity is no smaller than u_{jr} for each resource type r . Let e_j^{\max} be the maximal length of such an interval. In case of $e_j^{\max} < d_j$, job j can not be processed completely and the project can not be terminated within less than $LBC1 + d_j - e_j^{\max}$ periods. Hence, a lower bound LBC3 is defined as:

$$LBC3 = LBC1 + \max \left\{ 0, \max \left\{ d_j - e_j^{\max} \mid j \in J - CP \right\} \right\} \quad (4.2)$$

Example. Consider the partial schedule built by the critical path $CP = \langle 1, 3, 6, 8, 11, 12 \rangle$ of our example instance given in Figure 4.2. With $LBC1 = 16$, we obtain the time windows and values e_j^{\max} of the remaining jobs shown in Table 4.1. For example, consider job 7 which is incompatible to all jobs but job 6. Recall that a job i is said to be incompatible to another job j if both jobs can not be processed in parallel (cf. Definition 3.6, p. 76). Due to $ES_7 = 6$ and $LF_7 = 12$, we get $e_7^{\max} = 1$. Hence, executing job 7 in addition to the jobs on the critical path requires $d_7 - e_7^{\max} = 3$ further periods. The same is true for job 2 such that $LBC3 = 19$ is yielded.

Both, determining the critical path as well as computing the values e_j^{\max} , can be performed in $O(n^2)$ time. Since this is done consecutively, the overall complexity of LBC3 is $O(n^2)$, too. Two extensions of LBC3 have been proposed trying to consider several jobs, which can not be processed within their time windows, simultaneously:

- Demeulemeester (1992) determines a lower bound on the minimal project completion time necessary to schedule a further (partial) sequence of jobs along to those on a critical path. The sequence starts with a job h and ends with one of its (indirect) successors j . Both jobs are chosen such that they can not completely be processed within their time windows, i.e. $e_h^{\max} < d_h$ and $e_j^{\max} < d_j$. Additionally, the sequence contains all jobs building a single

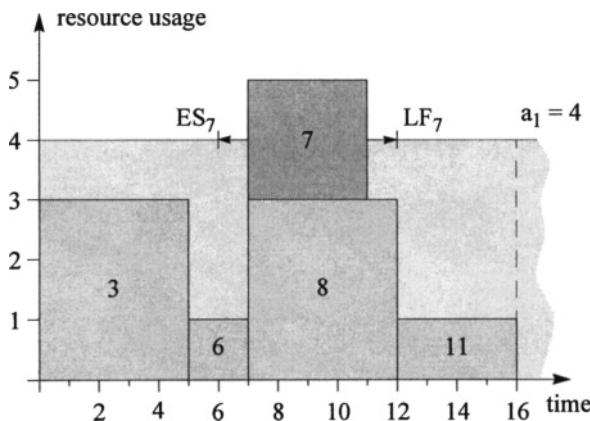


Figure 4.2. Computation of LBC3

path between h and j . The lower bound value can be obtained by a pseudo-polynomial dynamic programming approach and is calculated for all possible combinations of h and j .

- Brucker et al. (1996) consider a critical path as well as an additional disjoint path through the project network. Each of the two paths is interpreted as a task sequence of a job in a job shop problem with the objective of minimizing the completion time. Task pairs from the two jobs for which simultaneous processing causes a resource conflict in the original RCPSP instance must not be executed in parallel, whereas this is possible for all other pairs. Identifying such incompatible task pairs takes $O(m \cdot n^2)$ time. Subsequently, solving the special job-shop problem with two jobs, which gives a lower bound for RCPSP, has complexity $O(n \cdot \log n)$ (cf. Brucker (1988) and Domschke et al. (1997, section 5.2)). In order to improve on this bound, additional jobs which can still not be scheduled within their time windows are considered in an extended version (cf. Brucker et al. (1996)).

4.1.1.2 Bin Packing Bounds

RCPSP can be relaxed by discarding the precedence constraints and all capacity constraints but one for some resource type r . Then it is similar to single resource-constrained problems of the *bin packing type*. Therefore, lower bound arguments for such problems may be generalized (cf. Martello and Toth (1990), Labb   et al. (1991), Berger et al. (1992), and Scholl et al. (1997 a)).

Bin Packing Bound 1 (LBB1). For each resource type r at a time, the jobs are sorted and numbered according to non-increasing capacity requirements u_{jr} . Those jobs whose requirements exceed one half of the capacity supply ($u_{jr} > a_r / 2$) form a set $J_1(r)$. Since none of the jobs from $J_1(r)$ is compatible to any other one from this set, all of them can be scheduled sequentially in sorting order. The length t_1 of the resulting schedule is a lower bound on the project duration. This bound can be improved by considering a second set $J_2(r)$ consisting of all jobs with $u_{jr} \in (a_r / 3, a_r / 2]$. Note that due to the resource constraint for type r , at most one job $j \in J_2(r)$ can be executed in parallel to each job $i \in J_1(r)$. In order to find out which of these jobs can be processed concurrently to those from $J_1(r)$, the jobs from $J_2(r)$ are examined in the reverse sorting order. All jobs are scheduled as early as possible observing the resource constraint resulting in a schedule length t_2 . If $t_1 \geq t_2$ all jobs from $J_2(r)$ can be processed in parallel to jobs from $J_1(r)$ and $LBB1_r = t_1$ is a lower bound value with respect to resource type r . Otherwise, a project duration t_1 is not sufficient to perform all jobs from $J_1(r) \cup J_2(r)$. Then a lower bound is given by $\lceil (f_1 + f_2) / 2 \rceil$, where f_1 and f_2 are the finishing times of the two latest jobs within the schedule. Following this argumentation, we obtain a lower bound for resource type r by computing $LBB1_r = \max\{t_1, \lceil (f_1 + f_2) / 2 \rceil\}$.

Note that t_2 needs not be a lower bound on the project duration, because there may exist a sequence of the jobs from $J_2(r)$ which results in a shorter schedule. Therefore, the right-hand-side expression is restricted utilizing the fact that at most two of the jobs from $J_2(r)$ can be scheduled simultaneously.

The overall bound value $LBB1$ is taken as the maximum of all values $LBB1_r$ computed for the resource types $r = 1, \dots, m$. The computational complexity of $LBB1$ is $O(m \cdot n \cdot \log n)$, because each value $LBB1_r$ is easily computed in time $O(n)$ whereas the sorting can be realized in $O(n \cdot \log n)$ time, respectively.

Consider the following example with one resource type $m = 1$ and $a_1 = 24$. Precedence constraints are not relevant. The jobs are given in order of non-increasing resource usages u_{j1} and the dummy nodes 1 and 16 are omitted.

Table 4.2. Example for LBB1 and LBB2

job	2	3	4	5	6	7	8	9	10	11	12	13	14	15
d_j	2	3	6	4	2	7	4	3	5	7	3	9	8	6
u_{j1}	23	20	19	17	15	14	14	13	12	11	11	10	8	7

The result of scheduling the jobs 2 to 9 from $J_1(1)$ and 10 to 13 from $J_2(1)$ is given in Figure 4.3. The first set of jobs gives a schedule length of $t_1 = 31$ periods. Since only jobs 13 and 12 can be scheduled in parallel, an additional time of at least $\lceil (d_{10} + d_{11}) / 2 \rceil = 6$ is necessary. More formally, we obtain $LBB1 = 37$ due to $f_1 = 36$ and $f_2 = 38$.

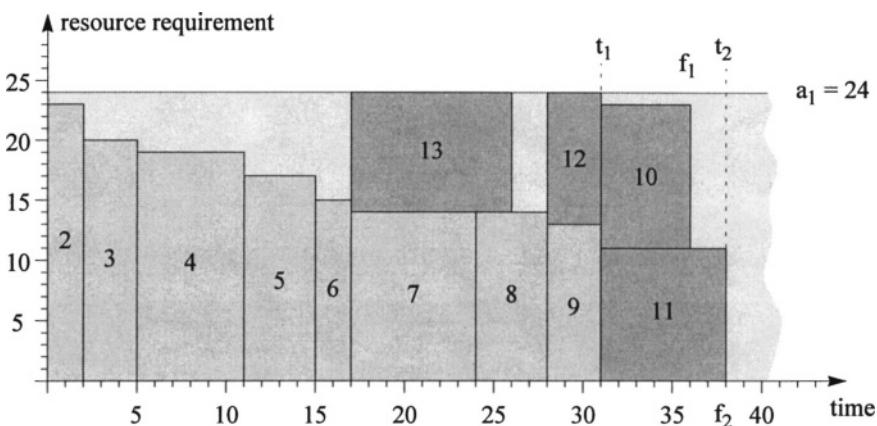


Figure 4.3. Computation of LBB1

Bin Packing Bound 2 (LBB2). According to LBB1, the precedence constraints and all capacity constraints except for one are omitted. Furthermore, the same sorting of jobs is used. However, this bound argument additionally considers the jobs with $u_{jr} \leq a_r / 3$.

Assume that one job j with $u_{jr} \leq a_r / 2$ and all jobs $h < j$, i.e., having the same or a larger capacity requirement of type r , have to be scheduled. Furthermore, let $p(j) = \max\{h \mid u_{hr} + u_{jr} > a_r\}$ be the number of the last job (in the sorting order) which can not be scheduled in parallel to any other job $h \leq j$. That is, the jobs $1, \dots, p(j)$ have to be performed exclusively, while among the remaining jobs $p(j)+1, \dots, j$ at most $q(j) = \lfloor a_r / u_{jr} \rfloor$ can be performed in parallel to each other. Therefore, the total duration of the jobs $1, \dots, p(j)$ and the total duration of the jobs $p(j)+1, \dots, j$ over $q(j)$ is a lower bound on the project duration. One may get an improved bound value on the time required for executing the set $S_j = \{p(j)+1, \dots, j\}$ through computing LBC2 (cf. Section 4.1.1.1) for this reduced problem concerning all resources (abbreviated by $LBC2(S_j)$).

Applying this logic to all jobs $j \in J - J_1(r)$, namely those with $u_{jr} \leq a_r / 2$, gives a lower bound $LBB2_r$ concerning the resource type r considered:

$$LBB2_r = \max \left\{ \sum_{h=1}^{p(j)} d_h + \max \{ LBC2(S_j), \left\lceil \sum_{h=p(j)+1}^j d_h / q(j) \right\rceil \} \mid j \in J - J_1(r) \right\} \quad (4.3)$$

Like with $LBB1$, a lower bound $LBB2_r$ is computed for each resource type $r = 1, \dots, m$ and the maximal value defines the overall bound $LBB2$. The computational complexity is $O(m \cdot n \cdot \log n)$, because $LBB2_r$ can be computed in linear time $O(n)$ given a sorted list of jobs for any r .

For our example with one resource type, we get $LBB2 = 39$ for $j = 15$ with $p(j) = 4$ and applying $LBC2$ to $S_{15} = \{5, \dots, 15\}$.

Remark 4.1. As the computational experiments in Section 7.3.1, pp. 275, show, the arguments $LBB1$ and $LBB2$ perform rather poor which is mainly due to considering only a single resource type. In general, this disadvantage may partially be overcome by aggregating the availability constraints for the different resource types using surrogate constraints (cf. Salkin (1975, section 7.5.3)). However, own preliminary computational results have revealed that even applying this rather evolved concept does not lead to improvements of $LBB1$ and $LBB2$ such that they can cope with arguments incorporating information on all resource types like the node packing bounds and the parallel machine bounds.

4.1.1.3 Node Packing Bounds

The following bound arguments basically consist of finding a subset S of jobs each of which is incompatible to any other one within this set (cf. Definition 3.6, p. 76). Due to this pairwise incompatibility, these jobs must be scheduled sequentially thereby defining a lower bound equal to the sum of their durations. The problem of determining a set S such that the lower bound value yielded is maximized can be formulated as a *weighted-node packing problem* which may be described as follows (cf. Mingozi et al. (1998)). Consider an undirected graph $\tilde{G} = (V, E, w)$ with a set of nodes V and a set of edges E connecting a subset of these nodes. The values w_j denote the weight of each node $j \in V$. The weighted-node packing problem now consists of finding a subset $S \subset V$ such that no pair of jobs $(i, j) \in S$ is connected by an edge within \tilde{G} and the sum of the weights w_j with $j \in S$ is maximized.

RCPSP may be relaxed to a weighted-node packing problem by introducing a node with $w_j = d_j$ in \tilde{G} for each job $j \in V$. Furthermore, an edge is defined for each pair of jobs (i, j) if this pair can be processed in parallel, i.e., no precedence relationship exists and $u_{ir} + u_{jr} \leq a_r$ for $r = 1, \dots, m$. However, solving the weighted-node packing problem is NP-hard (cf. Nemhauser and Wolsey (1988, chapter I.6)). This led to the development of the following bound arguments.

Node Packing Bound (LBN1). This argument which has been proposed by Mingozi et al. (1998) determines a heuristic solution for the weighted node packing problem defined as described above. First, the set of all non-dummy jobs is sorted and stored in an ordered list. Subsequently, the first job is removed from the list and added as first element to S . All jobs being compatible to this job are deleted from the list. This process is repeated until the list is empty. The total duration of the jobs in S defines a lower bound LBN1.

Different solutions can be obtained by applying several sorting criteria (cf. De meulemeester and Herroelen (1997 b) and Mingozi et al. (1998)). An obvious approach is to set the jobs of a critical path at the first positions followed by the remaining jobs in increasing order of the number of jobs which can be processed in parallel to the job considered. This rule reflects the fact that as many jobs as possible should remain in the list after each iteration. Additionally, the list is cyclically rotated by one position and the procedure is applied repeatedly. An improved version of the node packing bound does not completely delete from the list a job h being compatible to a job j just added to S (cf. De Reyck and Herroelen (1998 a)). Instead, if $d_h > d_j$, job h stays in the list with a modified duration of $d_h := d_h - d_j$. Since the list has to be processed once completely and updating the remaining jobs requires checking all resource constraints, the computational complexity of LBN1 is $O(m \cdot n^2)$, if rotation is omitted.

Example. For the example given in Figure 4.1, the corresponding list may be $L = \langle 3, 6, 8, 11, 4, 2, 7, 5, 9, 10 \rangle$ according to the sorting criterion mentioned above. The jobs 3, 6, 8, 11 of the critical path are successively included in S , because they are incompatible to each other. After considering job 3, job 5 is deleted. Job 6 requires reducing the duration of the jobs 2 and 7 to $d_2 = 1$ and $d_7 = 2$, respectively. Including the jobs 8 and 11 results in deleting the jobs 9 and 10. Now, only job 4 and the remainders of 2 and 7, which are incompatible to each other, are contained in the list. Therefore, they are included in S resulting in a bound $LBN1 = 16 + 1 + 1 + 2 = 20$. The original version of the node

packing bound, which considers complete jobs only, would give a lower bound value of 17.

Extended Node Packing Bound (LBN2). LBP1 can be improved by applying additional bound arguments to a residual RCPSP instance consisting of the jobs remaining in the list *after each iteration*, i.e., always just after including a job in S (cf. Klein and Scholl (1999)). These jobs are considered with their reduced durations. The corresponding precedence network is derived from the original one by deleting all nodes representing jobs removed from the list, preserving original direct and indirect precedence relations. The best bound value obtained for the residual instance may be added to the total duration of jobs currently in S thereby defining a lower bound on the initial instance in each iteration. The maximum of those values serves as LBN2.

To keep the computational effort low, we only use the simple bound arguments LBC1 and LBC2 for evaluating the residual projects. Due to applying LBC1 anyway, it is no longer appropriate to put the jobs of the critical path at the beginning of the list. Instead, jobs are sorted in increasing order of the number of jobs which may be processed in parallel as proposed by Demeulemeester and Herroelen (1997 b). Ties are broken according to longer job durations such that among a set of pairwise compatible jobs the longest one is preferred.

In each of $O(n)$ iterations, updating S and removing jobs from the list requires time $O(m \cdot n)$. Applying LBC1 and LBC2 to the residual project takes time $O(n^2)$ and $O(m)$, respectively. Thus, the total complexity is $O(n \cdot (m \cdot n + n^2))$, if rotation of the list is omitted.

Example. Using the adapted sorting criterion, we obtain the following list $L = \langle 3, 4, 11, 8, 7, 2, 6, 9, 10, 5 \rangle$ excluding the dummy jobs. After successively including the pairwise incompatible jobs 3, 4, 11, and 8 in S and removing 5, 9, and 10 from the list, we yield $\sum_{j \in S} d_j = 15$. For the residual instance which is defined by the remaining jobs 2, 6, and 7 with original durations, the critical path consists of the jobs 2 and 7 with a length of $LBC1 = 7$. Applying the capacity bound only gives $LBC2 = \lceil (3 \cdot 2 + 2 \cdot 1 + 4 \cdot 2) / 4 \rceil = 4$. Hence, a bound value of 22 is reached in this iteration. Since no better value is found in another iteration, LBN2 = 22.

Generalized Node Packing Bound (LBN3). We describe a further extension of the node packing bound LBN1 (cf. Klein and Scholl (1999)). In general, one may identify a subset S of all jobs any triplet of which is *incompatible*, i.e., no subset of three or more jobs can be scheduled in parallel, either due to precedence or resource constraints. Furthermore, some jobs from S (forming a subset S_1) may be incompatible to any other (single) job of S and must be performed exclusively. Then a lower bound on the project duration is defined as follows (LBC2(S) denotes the application of LBC2 to a subset S of jobs, (cf. Section 4.1.1.1)):

$$LBN3(S) = \sum_{j \in S_1} d_j + \max \left\{ LBC2(S - S_1), \left\lceil \left(\sum_{j \in S - S_1} d_j \right) / 2 \right\rceil \right\} \quad (4.4)$$

The set S may be determined heuristically as in case of LBN1 and LBN2. For example, each job may be given a priority value equal to the number of incompatible triplets in which it is contained. The jobs are considered in increasing order of these priority values. Ties are broken in favor of longer job durations. A job is added to S whenever it is incompatible to any pair of jobs already contained. If it is even incompatible to any single job of S , it also becomes a member of S_1 . Otherwise, its inclusion causes at least one job to be removed from S_1 (but not from S). Always after adding a job, the expression $LBN3(S)$ is evaluated and the maximum of all those values serves as lower bound $LBN3$. Due to checking triplets of jobs concerning m resource types, LBN3 requires $O(m \cdot n^3)$ time when the list is not rotated.

Example. Sorting the non-dummy jobs as mentioned above yields the following list $L = \langle 3, 8, 6, 4, 7, 11, 2, 5, 9, 10 \rangle$. After four iterations $S_1 = S = \{3, 8, 6, 4\}$ is obtained. Now, job 7 has to be added. Since it is compatible to 6, both are members of S but not of S_1 . The current lower bound value is $LBN3(S) = 11 + \max\{\lceil(2 \cdot 1 + 4 \cdot 2)/4\rceil, \lceil 6/2 \rceil\} = 14$. The overall maximum value $LBN3 := LBN3(S) = 15 + \max\{\lceil 16/4 \rceil, \lceil 9/2 \rceil\} = 20$ is obtained two iterations later for $S_1 = \{3, 8, 4, 11\}$ and $S = S_1 \cup \{6, 7, 2\}$. After including 5 and 9 and removing all jobs from S_1 , the process stops, because job 10 can not be added due to, e.g., the compatible triplet 2, 9, 10.

Remark 4.2. LBN3 may be extended. According to LBN1, jobs may be included in S or S_1 with reduced durations rather than deleting them completely. So, one part of a job may belong only to S and another one also to S_1 . In the

example, a value of $LBN3(S) = 18 + \max\{\lceil 8/4 \rceil, \lceil 5/2 \rceil\} = 21$ is yielded for $S = \{3, 8, 6, 4, 7, 11, 2\}$. The increase compared to above is due to the jobs 7 and 2, now partially contained in S_1 with durations 2 and 1, respectively. Additionally, LBC1 and LBC2 may be applied to an appropriately defined residual instance as done for LBN2. Beyond these extensions, LBN2 and LBN3 can be generalized by considering k -tuples ($k \geq 4$) of jobs, which can not be processed simultaneously. However, the computational complexity of this extended approach is $O(m \cdot n^k)$. Instead, in order to avoid such high computational effort, all but one capacity constraint can be discarded. Then, different sets S could be determined like in the case of LBM1 (see below).

4.1.1.4 Parallel Machine Bounds

These bound arguments are based on the relaxation of RCPSP to a special *parallel-machine scheduling problem* (cf. Carlier and Latapie (1991)) which can be described as follows: A set S of jobs has to be performed on k identical parallel machines with at most one job being processed on each machine at a time. At the beginning of the planning horizon, not all of the jobs are available immediately. This may, e.g., be due to preprocessing or restricted availability of materials. The amount of time which has to pass before a job j becomes available on the machines is called *head* α_j of j . Analogously, terminating a job j may require additional time after leaving the machines, e.g., for postprocessing, called *tail* ω_j of j . The objective of the parallel-machine problem is to find a sequence of all jobs for which the *makespan*, i.e., the amount of time necessary for terminating all jobs, is minimized.

A subset of jobs of the original RCPSP instance for which it is known that at most k jobs of the subset can be performed at a time may be interpreted as a set S of jobs to be scheduled on k parallel machines. Additionally, heads and tails for those jobs may be derived using the information inherent to earliest and latest starting and finishing times. Since ES_j is a lower bound on the starting time of job j in the RCPSP instance, j can not be available earlier on a "machine". Hence, $\alpha_j = ES_j$ for all $j \in S$. Accordingly, tails ω_j may be determined. At least $\omega_j = LF_n - LF_j$ periods have to pass after performing j before the whole project is completed. By analogy, having processed job j on one of the "machines", a tail of ω_j periods is necessary for its termination.

Exactly solving a parallel-machine problem defined by a set S of jobs yields a lower bound on the completion time of the underlying RCPSP instance. Since the parallel-machine problem is NP-hard, the effort may usually be too high (cf. Karp (1972) and Domschke et al. (1997, section 5.1.4.3)). Nevertheless, bound arguments on hand for the parallel-machine problem can be applied.

Parallel-Machine Bound (LBM1). To determine a set S of jobs for a given value of k , Carlier and Latapie (1991) discard all but one resource type r . Since for each type r several subsets of jobs defining a parallel machine problem may exist, they present an approach for enumerating all of them, which results in a rather high total computational effort. Therefore, we describe a restricted approach to determine a *single* subset S for each resource type r : S contains all jobs j with their demand u_{jr} not being less than $u_r^{\min} = \lfloor a_r / (k+1) \rfloor + 1$, because at most k of them can be processed in parallel. Performing a job j requires at least $\beta_j = \lfloor u_{jr} / u_r^{\min} \rfloor$ machines. Therefore, β_j copies of j are included in S , i.e., j is replicated $\beta_j - 1$ times.

For $S' \subseteq S$ being an arbitrary subset of S , a lower bound on the makespan can be calculated by (cf. Klein and Scholl (1999)):

$$M(S') = \min \{ \alpha_j \mid j \in S' \} + \left\lceil \sum_{j \in S'} d_j / k \right\rceil + \min \{ \omega_j \mid j \in S' \} \quad (4.5)$$

The maximum of all those values $M(S')$ for any $S' \subseteq S$ defines a lower bound on RCPSP. It can be computed in $O(n^3)$ by systematically considering all pairs of values α_h and ω_i with $h, i \in S$ and $\alpha_h \leq \alpha_i$, $\omega_h \geq \omega_i$. The (maximal) set S' corresponding to α_h and ω_i being the minimum head and tail in (4.5) contains all jobs $j \in S$ (including h and i) which fulfill $\alpha_h \leq \alpha_j$ and $\omega_j \geq \omega_i$. The computational effort can be reduced to $O(n^2)$ by considering the pairs (α_h, ω_i) following a lexicographic ordering. Then the sets S' and the corresponding sums of job durations are derived from the preceding pair by a constant time update. An alternative and more complex method of computing S' is presented in Carlier and Latapie (1991).

For each resource type r , a set S is constructed as described above and the maximal $M(S')$ with $S' \subseteq S$ is computed. The maximum of all bound values obtained in this manner (for $k = 1, 2, \dots$ and $r = 1, \dots, m$) serves as LBM1.

Example. Applying a forward and backward pass to our project of Figure 4.1 results in the heads and tails of jobs given in Table 4.3 (cf. Table 4.1). If we

Table 4.3. Heads and tails

jobs	2	3	4	5	6	7	8	9	10	11
$\alpha_j = ES_j$	0	0	0	3	5	6	7	7	7	12
LF_j	5	5	5	8	7	12	12	16	16	16
$\omega_j = LF_n - LF_j$	11	11	11	8	9	4	4	0	0	0

consider $k = 1$ (one-machine problem) for the single resource type $r = 1$, we get $u_r^{\min} = 3$ and $S = \{3, 4, 8\}$. Evaluating (4.5) for all subsets of S gives the bound values $M(S' = \{3, 4\}) = 0 + 5 + 1 + 11 = 17$ and $M(\{3, 4, 8\}) = 15$. For $k = 2$, we obtain $u_r^{\min} = 2$ and $S = \{3, 4, 8, 7, 2\}$ with all jobs requiring at least $\beta_j \geq 1$ machine. The maximal bound value is determined by $M(\{2, 3, 4\}) = 0 + \lceil 9/2 \rceil + 11 = 16$. For $k = 3$, we again obtain $u_r^{\min} = 2$ and $S = \{3, 4, 8, 7, 2\}$ yielding the weaker value $M(\{2, 3, 4\}) = 0 + \lceil 9/3 \rceil + 11 = 14$. In case of $k = 4$, we have $u_r^{\min} = 1$ so that each job is contained in S in $\beta_j = u_{jr}$ copies. Hence, $M(S) = 0 + \left\lceil \sum_j u_{jr} \cdot d_j / a_r \right\rceil + 0 = 17$. This value is maximal, because no subset of S gives a better value in (4.5). The best overall bound value is LBM1 = 17.

Remark 4.3. For each value of k , the computational complexity of LBM1 is $O(m \cdot n^2)$. As indicated by the example in case of $k = 4$, LBM1 always gives results at least as good as LBC2 when $k = a_r$ is examined for each resource type r . Furthermore, it can be seen that it may be helpful, in particular when a_r is small, to determine the numbers β_j of copies more carefully, because, e.g., for $k = 2$ the jobs with $u_{jr} = 3$ actually could be duplicated.

One-Machine Bound (LBM2). Within the framework of LBN1, a weighted-node packing problem is solved in order to find a subset S of jobs each pair of which is incompatible. Since at most one job out of this set can be performed at a time, a corresponding *one-machine problem* can be defined which is a special case of the parallel-machine problem ($k = 1$) considered by LBM1. Following this idea, LBM2 proceeds as follows (cf. Klein and Scholl (1999)). A subset S of jobs is determined heuristically according to LBN1. In each iteration of constructing S , formula (4.5) is applied for $S' = S$, yielding a lower bound $M(S)$. Note that (4.5) can easily be evaluated by updating its expressions in constant time. Therefore, the complete process takes time $O(m \cdot n^2)$.

Since LBM2, in contrast to LBN1, considers heads and tails, a more suitable sorting criterion for the priority list is used. For each job j , the center of its time

window is computed by $CT_j = (ES_j + LF_j)/2$. Starting with an arbitrary job q , the jobs are added to S in order of increasing absolute distances of their centers CT_j to CT_q . Ties are broken in favor of the smallest number of jobs which can be processed in parallel to the job considered. This sorting rule follows the idea that it is advantageous to select jobs for S which have similar values of heads and tails, respectively. In this case, the sum of the minimum terms of (4.5) tends to become large. The maximum of the values $M(S)$ obtained during constructing S defines a lower bound $LBM2(q)$.

Example. We consider $q = 3$ with $ES_3 = 0$, $LF_3 = 5$ and $CT_3 = 2.5$ (cf. Table 4.1, p. 115). Applying the sorting criterion as described above results in the list $L = \langle 3, 4, 2, 5, 6, 7, 8, 9, 10, 11 \rangle$ excluding the dummy jobs. Following the usual process of constructing S we add the jobs 3, 4, 2 thereby discarding 5 and 6 and reducing the durations of 9 and 10 to 3 and 1, respectively. The jobs $j \in \{3, 4, 2\}$ may start immediately at time $\alpha_j = 0$. Since all those jobs have an identical tail of $\omega_j = 11$, we obtain $M(\{3, 4, 2\}) = 0 + 9 + 11 = 20$. Performing another two iterations, we yield $M(\{3, 4, 2, 7, 8\}) = 0 + 18 + 4 = 22$ which is the maximal value determined defining $LBM2(q=3)$.

In order to construct different sets S , each job in turn is used as the start job q yielding an overall lower bound $LBM2 = \max \{LBM2(q) \mid q = 2, \dots, n-1\}$. In the example, $LBM2 = 22$.

Two-Machine Bound (LBM3). It is obvious, that LBN3 and LBM1 may be combined in the same fashion as described for LBM2 in the previous section (cf. Klein and Scholl (1999)). In this case, a *parallel-machine problem* with $k = 2$ is present, because at most two jobs may be processed simultaneously. The set of jobs to be considered is determined as follows. Starting with an arbitrary job q , the heuristic of LBN3 is applied combined with the sorting criterion of LBM2. This results in a set S and a subset S_1 which contains all jobs being incompatible to all other jobs in S . As is the case with LBM2, formula (4.5) is evaluated always after adding a job to S . For this purpose, a set S' is formed which contains all jobs from S and a copy of each job currently in S_1 . Following Remark 4.2, a job may be part of S_1 with a reduced duration. Then, only this part is duplicated. By analogy with LBM2, the bound LBM3 is determined as the maximum value found starting with each job as q once. Each of those iterations takes time $O(m \cdot n^3)$.

Example. The jobs are added to S according to the list described for LBM2 starting with $q = 3$. In the fifth iteration, $S = \{3, 4, 2, 5, 6\}$. The subset S_1 consists of a part of job 3 with reduced duration of 2 and a part of job 2 with reduced duration of 1. The remainders of those jobs have been removed from this set due to adding the compatible jobs 5 and 6, respectively. The set S' for applying (4.5) is composed of all jobs in S and additional copies of the jobs 3 and 2 with reduced durations. We get $M(\{3, 2, 3', 2', 4, 5, 6\}) = 0 + \lceil 17/2 \rceil + 8 = 17$. After assigning the jobs 7 and 8, which additionally become members of S_1 with reduced durations of 2, respectively, $M(\{3, 4, 2, 3', 2', 5, 6, 7, 8, 7', 8'\}) = 0 + \lceil 30/2 \rceil + 4 = 19$ defining the maximal bound value for $q = 3$ is found. Since no better bound value is determined for other jobs q , LBM3 = 19.

4.1.1.5 Precedence Bounds

We propose rather simple but powerful bound arguments which also rely on examining incompatible pairs and triplets of jobs.

Precedence Bound 1 (LBP1). This bound argument is based on considering incompatible pairs of jobs. For any such pair (h, j) either a precedence relation exists or their cumulated capacity requirement exceeds the supply of any resource type r . In the latter case, a *disjunctive precedence relationship* can be defined, because h either must be finished before j can be started or vice versa (cf. Balas (1970)).

Therefore, lower bound values can be computed by testing both directions of the disjunctive precedence relationship between any incompatible pair of jobs $(h, j) \in IP$ (cf. Definition 3.6, p. 76). Consider such an incompatible pair (h, j) . First a directed arc is temporarily introduced from h to j thereby defining a test project $TP(h, j)$. To this test project any bound argument for RCPSP can be applied to compute a lower bound $LB(h, j)$ on the project duration given that h is performed before j . Analogously, a second test project $TP(j, h)$ is built by introducing a directed arc from j to h and a lower bound $LB(j, h)$ is determined. The lower bound on the original RCPSP instance is given by the minimum of $LB(h, j)$ and $LB(j, h)$. Then, a lower bound $LBP1$ is obtained by applying this test to all incompatible pairs, computing a lower bound value for the respectively resulting two test projects, and, finally, taking the maximal value found for any incompatible pair. That is, $LBP1$ is calculated as follows:

$$LBP1 = \max\{\min\{LB(h, j), LB(j, h)\} \mid (h, j) \in IP \text{ and } (h, j), (j, h) \notin A^*\} \quad (4.6)$$

However, in order to restrict the computational effort required for computing the lower bound value for any test project, we only apply the critical path bound LBC1 (cf. Section 4.1.1.1).

Example. Consider the incompatible job pair 7 and 8. Introducing the directed arc $(7,8)$ and applying LBC1 results in $\text{LB}(7,8) = 19$, while the reverse arc $(8,7)$ results in $\text{LB}(8,7) = 20$. Since evaluating the other incompatible pairs does not give improved bounds, $\text{LBP1} := \min\{\text{LB}(7,8), \text{LB}(8,7)\} = 19$.

Precedence Bound 2 (LBP2). LBP2 is an extension of LBP1 to incompatible triplets of jobs. If no pair of such jobs is incompatible, the resource conflict can be resolved by adding a single directed arc between one pair at a time, in total defining six test projects (three pairs, two directions). Otherwise, disjunctive precedence relations between the jobs of each resource-incompatible pair have to be directed, whereas existing precedence relations are preserved. Due to six sequences of three jobs being possible, up to six test projects have to be considered. These test projects are evaluated by applying a bound procedure, giving a lower bound for each of them. Among the values obtained, the minimum represents a lower bound on the original RCPSP instance. Examining all triplets of incompatible jobs and taking the maximal value yields the lower bound LBP2.

Example. Consider the triplet $(2,3,4)$. Each pair of those jobs is incompatible due to resource constraints. For each of the six test projects, representing all possible sequences, $\text{LBC1} = 20$. Since none of the other triplets yields an improved bound, $\text{LBP2} = 20$.

Remark 4.4. If the evaluation of test projects is restricted to computing LBC1, LBP1 and LBP2 can be calculated efficiently using heads and tails. Consider an incompatible pair (h,j) . The length of the longest path using arc (h,j) is equal to $\alpha_h + d_h + d_j + \omega_j$. If this length exceeds LBC1 for the initial project, introducing (h,j) leads to a new critical path in $\text{TP}(h,j)$ containing the jobs h and j . Sequences for incompatible triplets can be evaluated in the same fashion. Since these computations take constant time, LBP1 can be computed in $O(m \cdot n^2)$ time, which is needed for determining all incompatible pairs. The time complexity of LBP2 is $O(m \cdot n^3)$.

4.1.2 Complex Bound Arguments

In this section, we describe three different complex approaches for computing lower bounds. Though in particular the application of the approach described in

Section 4.1.2.3 yields good lower bound values, the computational effort for computing these values is rather large. Therefore, these arguments can not be incorporated into exact solution procedures except for, eventually, calculating an initial bound value before starting a procedure.

4.1.2.1 LP-Relaxation with Cutting Planes

In Section 3.2.1, different mathematical formulations for RCPSP using binary and integer variables have been presented. By relaxing the integrality restrictions for these variables, linear programming problems are received which can be solved in polynomial time. Christofides et al. (1987) have examined the effectiveness of choosing this type of relaxation for formulation 1 presented in Section 3.2.1.2, pp. 79. For this purpose, they replace the restrictions (3.12) ($x_{jt} \in \{0, 1\}$) by restrictions $x_{jt} \geq 0$ and $x_{jt} \leq 1$ for $j \in J$ and $t \in [1, \dots, \bar{T}]$, respectively. Furthermore, they modify the remaining restrictions and introduce cutting planes in order to yield sharper bound values.

The first modification ensures that the dummy end job n is not completed before one of its predecessor jobs which is possible after dropping the integrality constraints. For the example of Figure 4.1, p. 115 with $n = 12$, a solution vector \mathbf{x} of the LP-relaxation may, e.g., contain the entries $x_{11,16} = 0.9$, $x_{11,18} = 0.1$, and $x_{12,17} = 1$ for the jobs 11 and 12 without violating the precedence constraints (3.10) due to $17 \cdot 1 - (16 \cdot 0.9 + 18 \cdot 0.1) = 0.8 > 0$. This is avoided by replacing a part of the original resource constraints (3.11) with the following restrictions:

$$\sum_{j \in E(t)} u_{jr} \cdot \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_j-1, LF_j\}} x_{jq} \leq a_r \cdot \left(1 - \sum_{q=EF_n}^{t-1} x_{nq} \right)$$

for $r \in R$ and $t \in [EF_n + 1, \dots, \bar{T}]$ (4.7)

If job n is terminated at the end of period $t-1$ ($\sum_{q=EF_n}^{t-1} x_{nt} = 1$), the restrictions (4.7) guarantee that no resources are available for processing other jobs in period t as well as in the subsequent ones $q = t+1, \dots, \bar{T}$. In the above example with $x_{12,17} = 1$, this is true for the periods 18, Note that it is not sufficient to modify only the restrictions of a single resource type, because not all jobs may require this type for their execution.

Of course, the same problem of partially ignoring precedence relationships may arise for any pair of jobs $(i,j) \in A$. Considering our example, solving the LP-relaxation may, e.g., yield $x_{35} = 0.8$, $x_{39} = 0.2$, and $x_{68} = 1$ for job 3 and its successor 6. Again, this solution does not hurt the precedence constraints (3.10) because of $(8-2) \cdot 1 - (0.8 \cdot 5 + 0.2 \cdot 9) = 0.2 > 0$. Therefore, the precedence constraints may be reinforced by additionally introducing the following cutting planes:

$$\sum_{\substack{q=t \\ q=EF_j}}^{LF_i} x_{iq} + \sum_{q=t}^{t+d_j-1} x_{jq} \leq 1 \quad \text{for } j \in J, i \in P_j, t \in [ES_j + 1, \dots, LS_j] \quad (4.8)$$

The restrictions (4.8) ensure that if job j is (fully) terminated until the period $t + d_j - 1$, none of its predecessors i can be (partially) completed in period t or later.

Another effect of relaxing the integrality restrictions consists in the resource constraints becoming very loose. With a resource availability of $a_1 = 4$, the jobs 2 and 3 must not be processed in parallel due to $u_{21} = 2$ and $u_{31} = 3$. However, solving the LP-relaxation may, e.g., result in $x_{25} = 0.5$, $x_{28} = 0.5$, $x_{35} = 0.6$, and $x_{3,10} = 0.4$. Hence, for the periods 3, 4, and 5 the total resource demand of the jobs 2 and 3 is $0.5 \cdot 2 + 0.6 \cdot 3 = 2.8$ such that no resource conflict seems to occur. The same is true for the following periods 6, 7, and 8 with $0.5 \cdot 2 + 0.4 \cdot 3 = 3.2$. To partially overcome this problem, Christofides et al. (1987) propose the use of the following cutting planes:

$$\sum_{j \in S \cap E(t)} x_{jt} \leq 1 \quad \text{for } S \in IPS \text{ and } t \in [1, \dots, \bar{T}] \quad (4.9)$$

According to the node packing bounds described in Section 4.1.1.3, S denotes a set of pairwise incompatible jobs. Among those jobs, at most one can be terminated at period t . Unfortunately, the maximum number of these sets is of the order $O(2^{n-2})$ such that determining all of them is prohibitive. Instead, several sets forming the collection IPS can be selected arbitrarily. The most simple form consists in restricting to the incompatible pairs of jobs contained in IP which can be computed in $O(m \cdot n^2)$ time. Another possibility consists in applying the heuristic described for LBN1 with different sorting criteria. Surprisingly, no proposal of determining the sets S is contained in Christofides et al. (1987).

The last modification concerns the objective function in order to avoid that the termination of the dummy end job n is "scattered" over several periods. In our example, this effect may, e.g., occur due to a solution with $x_{12,16} = 0.9$ and $x_{12,18} = 0.1$ yielding a smaller objective function value ($CT(\mathbf{x}) = 16.2$) than a solution with $x_{12,17} = 1$. In order to reduce this effect, the following modified objective function may be used:

$$\text{Minimize } CT(\mathbf{x}) = \sum_{\substack{t=EF_n \\ t=LF_n}}^{LF_n} f(t) \cdot x_{nt} \quad (4.10)$$

If $f(t)$ is a strongly monotonous function of t , both the modified (unrelaxed) problem P' as well as the (unrelaxed) original one P are equivalent, i.e., an optimal solution for P' is also optimal for P . Therefore, a solution obtained solving the LP-relaxation of P' may be put into the original objective function yielding a valid lower bound for P . For the LP-relaxation, choosing a convex function $f(t)$ with $f(t)$ increasing exponentially in t results in minimizing the latest period and, hence, the number of periods in which job n is partially terminated. Christofides et al. (1987) have examined the function $f(t) = 10^t$. For this function, the solution with $x_{12,17} = 1$ is clearly preferred to the one with $x_{12,16} = 0.9$ and $x_{12,18} = 0.1$.

4.1.2.2 Lagrangean Relaxation

Besides the LP-relaxation, the Lagrangean relaxation provides another common technique for computing lower bounds on combinatorial optimization problems (cf. Fisher (1973, 1981), Geoffrion (1974), and Domschke (1997, chapter 3.4)).

The basic rationale of Lagrangean relaxation consists of removing a set of constraints from the original problem and incorporating them into the objective function such that the resulting problem is easier to solve than the original one. For this purpose, the weighted negative slacks of the removed constraints are added to the objective function. Most commonly, the according weights are referred to as *Lagrangean multipliers*. Choosing non-negative multipliers arbitrarily and solving the so-defined problem yields a valid lower bound on the optimal objective function value of the original (minimization) problem.

The major task of applying Lagrangean relaxation is to find a set of multipliers such that the bound value obtained is as large as possible. Most often this is achieved by applying *ascent methods* (cf. Held et al. (1974) and Domschke (1997, section 3.4.3)). In each iteration of such methods, a solution for the relaxed problem is computed using the current set of multipliers. Subsequently, based on the solution obtained, the multipliers are modified such that, hopefully, in the next iteration a solution with a larger objective function value and, hence, a sharper bound value is obtained. This process is continued until a stopping criterion is fulfilled, e.g., a prespecified number of iterations have been performed without finding an improved bound value.

A Lagrangean relaxation based bound argument for RCPSP based on formulation 1 (cf. Section 3.2.1.2, pp. 79) has been proposed by Christofides et al. (1987). The resource constraints (3.11) are discarded from the problem and integrated into the objective function using multipliers μ_{rt} for each restriction with $r = 1, \dots, m$ and $t = 1, \dots, \bar{T}$. This results in the following problem which has to be solved in order to obtain a lower bound value for a given set of Lagrangean multipliers:

$$\text{Minimize } CT(x) = \sum_{t=EF_n}^{LF_n} t \cdot x_{nt} + \sum_{r=1}^m \sum_{t=1}^{\bar{T}} \mu_{rt} \cdot \left(\sum_{j \in E(t)} u_{jr} \cdot \sum_{q=\max\{t, EF_j\}}^{\min\{t+d_j-1, LF_j\}} x_{jq} - a_r \right) \quad (4.11)$$

subject to (3.9), (3.10), (3.12)

In order to determine Lagrangean multipliers yielding a sharp lower bound value, Christofides et al. (1987) apply the ascent method described by Held et al. (1974). In each iteration of the ascent method, a solution for the problem defined by the current multipliers is determined by a branch and bound procedure. Most recently, Drexl and Kimms (1998 a) showed that the above problem can be transferred into a *total weighted completion time problem*. For solving this problem, they introduce a dynamic programming approach which requires $O(n^2 \cdot \bar{T} \cdot \max\{n, \bar{T}\})$ time.

4.1.2.3 Set Covering Based Approach

The following approach for computing lower bounds for RCPSP has been introduced by Mingozi et al. (1998). It is based on the formulation 6 presented in Section 3.2.1.7, pp. 87. By relaxing this formulation, a generalized set covering problem may be obtained the LP-relaxation of which can be solved to yield a lower bound value for RCPSP (cf. Nemhauser and Wolsey (1988, pp. 6), Domschke and Drexl (1996, section 4.2.5), and Domschke (1997, section 5.4)).

After discarding the precedence constraints (3.58) in formulation 6 and allowing for preemption of jobs, the variables x_{jt} can be eliminated from the formulation completely by omitting the restrictions (3.55) to (3.57), and (3.58) and replacing the objective function. This leads to the following optimization problem (4.12)-(4.15):

$$\text{Minimize } CT(\mathbf{p}) = \sum_{h=1}^q \sum_{t=1}^{\bar{T}} p_{ht} \quad (4.12)$$

subject to

$$\sum_{h=1}^q p_{ht} \leq 1 \quad \text{for } t \in [1, \dots, \bar{T}] \quad (4.13)$$

$$\sum_{h \in CS_j} \sum_{t=ES_j+1}^{LF_j} p_{ht} = d_j \quad \text{for } j \in J \quad (4.14)$$

$$p_{ht} \in \{0, 1\} \quad \text{for } h = 1, \dots, q \text{ and } t = 1, \dots, \bar{T} \quad (4.15)$$

Note that (4.12) represents an alternative objective function for formulation 6, because the smallest project completion time is determined by minimizing the number of periods in which a set $S_h \in CS$ is processed. The next relaxation step consists of dropping the assignment of compatible sets to periods such that the restrictions (4.13) become superfluous. Furthermore, the formulation may be simplified by introducing variables $v_h = \sum_{t=1}^{\bar{T}} p_{ht}$ for $h = 1, \dots, q$:

$$\text{Minimize } CT(\mathbf{v}) = \sum_{h=1}^q v_h \quad (4.16)$$

subject to

$$\sum_{h=1}^q v_h = d_j \quad \text{for } j \in J \quad (4.17)$$

$$v_h \geq 0 \text{ and integral} \quad \text{for } h = 1, \dots, q \quad (4.18)$$

Though the above formulation is much more simple than the original one, it still requires determining all compatible sets out of CS. To reduce the computational effort, only the *maximal sets* may be considered to which no further job can be added without the respective set becoming incompatible. With the index set $Q = \{h \mid S_h \in CS \text{ and no } S_{h'} \in CS \text{ with } S_h \subset S_{h'} \text{ exists}\}$, we finally obtain the following generalized set covering problem:

$$\text{Minimize } CT(\mathbf{v}) = \sum_{h \in Q} v_h \quad (4.19)$$

subject to

$$\sum_{h \in Q} v_h \geq d_j \quad \text{for } j \in J \quad (4.20)$$

$$v_h \geq 0 \text{ and integral} \quad \text{for } h \in Q \quad (4.21)$$

Note that the equations (4.17) have to be replaced by the inequalities (4.20) due to discarding non-maximal compatible sets. Nevertheless, both formulations yield solutions with identical objective function values, because possibly replacing a set $S_{h'}$ contained in a solution for (4.16)-(4.18) by a set $S_h \supset S_{h'}$ does not change the total number of sets required (cf. Mingozzi et al. (1998)). The main difference between the generalized set covering problem defined by (4.19)-(4.21) and the standard one consists in the circumstance that the standard one only considers the case $d_j = 1$ for all jobs (items) $j \in J$.

As a generalization of the set covering problem, the problem modeled by (4.19)-(4.21) is NP-hard (cf. Nemhauser and Wolsey (1988, chapter I.6)). Therefore, Mingozzi et al. (1998) restrict to determining all maximal compati-

ble sets by systematic enumeration and computing the optimal solution of the LP-relaxation of (4.19)-(4.21) in order to obtain a lower bound for RCPSP. A corresponding algorithm using the revised simplex method is described in Weglarz et al. (1977). However, though only maximal compatible sets are determined, their number may soon become too large to store all of them even on well-equipped computer systems. For this reason, Baar et al. (1998) have developed an approach based on column generation which generates only those sets required for calculating the optimal solution of the LP-relaxation.

Remark 4.5. The computational results in Section 7.3.4, pp. 284, show that the set covering approach is among the most competitive procedures for computing lower bounds on RCPSP, though the precedence relationships are discarded and preemption is allowed. This is partially due to the fact that the precedence relationships are implicitly considered when computing the incompatible sets.

4.2 Destructive Improvement

In this section, the meta-strategy *destructive improvement* for computing lower bounds on combinatorial minimization problems is described which has been developed by Klein and Scholl (1999). In Section 4.2.1, it is compared to other meta-strategies developed for this purpose and explained by means of a simple example. Its application to RCPSP is demonstrated in Section 4.2.2.

Though this technique is rather new, there already exist a few successful applications. Brucker and Knust (1998, 1999 b) have combined it with the set covering based approach described in Section 4.1.2.3 (cf. Section 7.3.4, pp. 284). Heilmann and Schwindt (1997) and Schwindt (1998) have applied the destructive improvement technique to RCPSP with maximum time lags, clearly providing the best lower bound arguments for this problem. Finally, Brucker and Knust (1999 a) show how destructive improvement can be used to compute lower bounds for a job-shop problem considering transportation times between different shops.

4.2.1 Meta-Strategies for Computing Lower Bounds

Traditionally, lower bounds for minimization problems are computed by relaxing the problem considered and solving the relaxation exactly (or solving the dual of the relaxation heuristically). Usually, the most complicating restrictions

are relaxed or even omitted (cf. Section 4.1 containing different examples for RCPSP). By relaxing complicating restrictions important information about the special problem structure often get lost such that bounds may be rather weak. Therefore, *Lagrangian relaxation* methods incorporate restrictions into the objective function by penalizing violations such that as many as possible of those constraints can be fulfilled within the solution of the relaxation (cf. Section 4.1.2.2). Another approach, which is called *additive bounding*, successively solves different relaxations of the problem and combines results via the concept of reduced costs (see, e.g., Fischetti and Toth (1989)). However, such bound computations are usually expensive and success in achieving strong bounds is questionable. These traditional techniques of computing lower bounds are referred to as *constructive or direct methods* because they directly tackle the problem by constructing solutions to relaxed problems.

Within *destructive improvement*, lower bounds are obtained by thoroughly examining trial bound values LB. For example, one may consider a bound value LB for a problem P determined by any constructive method. The value LB is utilized to restrict the set of feasible solutions by setting LB as the maximal value of the objective function allowed. If this restricted set of feasible solutions is empty, the objective function value of the optimal solution of P must exceed the value LB. Therefore, $LB + 1$ is a valid lower bound (provided that the objective function values are integral). In order to find out whether a feasible solution with its objective function value not exceeding LB may exist or not, two steps are performed.

To be more specific, let us consider a general formulation of problem P as an integer linear program where the linear objective function $f(\mathbf{x})$ is restricted to integer values whenever the vector \mathbf{x} is integral:

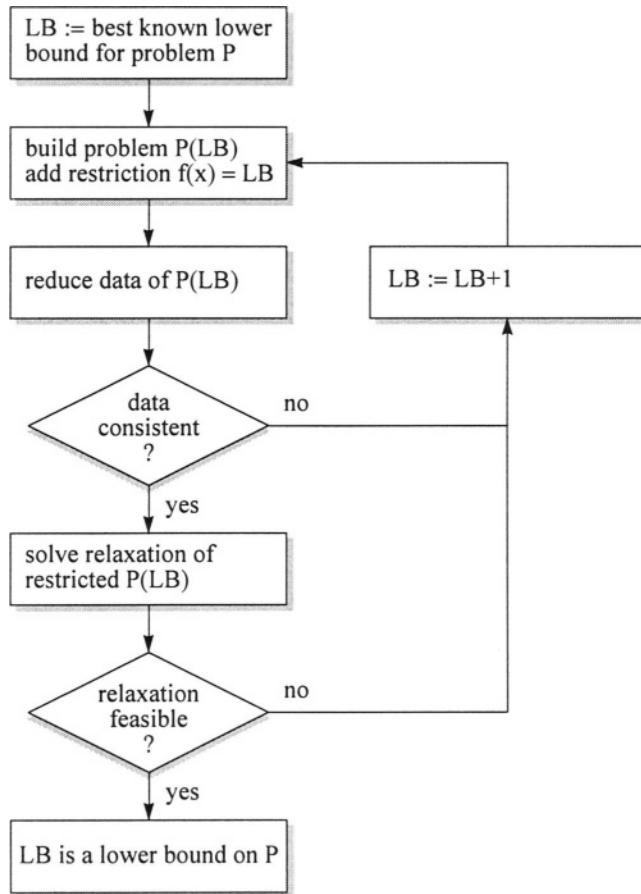
$$\text{Minimize } f(\mathbf{x}) \quad (4.22)$$

subject to

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \text{ and integer} \quad (4.23)$$

A *restricted problem* P(LB) is obtained by additionally enforcing $f(\mathbf{x})$ to be equal to LB:

$$f(\mathbf{x}) = LB \quad (4.24)$$

**Figure 4.4.** Destructive improvement

The *first step* consists of applying *reduction techniques* to the restricted problem $P(LB)$ in order to find additional constraints (cutting planes) or to modify problem parameters so as to strengthen the problem data. Possibly, a contradiction occurs and feasibility of $P(LB)$ is precluded. Otherwise, in a *second step* the restricted and strengthened problem is relaxed and solved, e.g. by applying a constructive bound argument. If the relaxation has no feasible solution, $P(LB)$ is infeasible, too, and LB is no valid objective function value. If neither step is able to exclude the possible feasibility of $P(LB)$ the process stops and LB serves as lower bound value. (Sometimes, a feasible solution of the relaxation is even feasible for P and the process stops with an optimum.) Otherwise, LB is increased by 1 and the two steps are repeated.

Remark 4.6. The problems $P(LB)$ are decisions problems and thus do not require an objective function (cf. Section 3.2.3, pp. 92).

The principle of destructive improvement just outlined is illustrated in Figure 4.4. In order to demonstrate its application, the following example problem P is considered:

$$\text{Minimize } f(\mathbf{x}) = 3 \cdot x_1 + 2 \cdot x_2 \quad (4.25)$$

subject to

$$9 \cdot x_1 + x_2 \geq 25 \quad (4.26)$$

$$4 \cdot x_1 + 7 \cdot x_2 \geq 50 \quad (4.27)$$

$$x_1, x_2 \geq 0 \text{ and integer} \quad (4.28)$$

Traditionally, the LP-relaxation of P is built by dropping the integrality constraints of the variables and solved by some LP-solver. The optimal LP-solution of the example is $\mathbf{x}^* = (2.12, 5.93)$ with $f(\mathbf{x}^*) = 18.22$ (cf. Figure 4.5). Since the cost coefficients in the objective function are integral, $LB = \lceil 18.22 \rceil = 19$ is a first lower bound on the optimal objective function value.

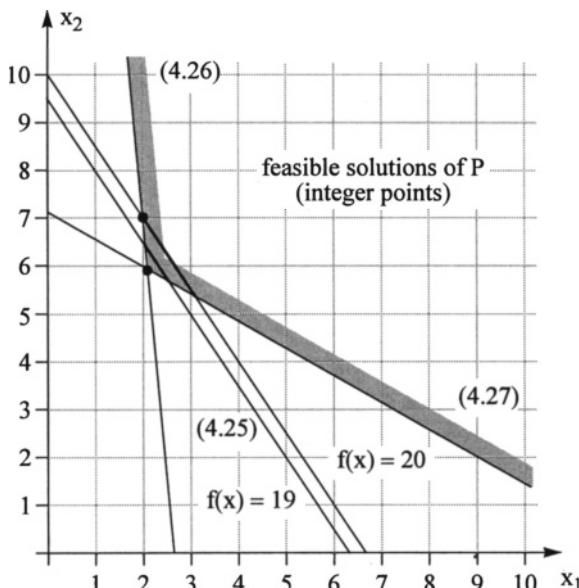


Figure 4.5. Problems P and $P(LB)$

The destructive improvement method starts with this value of LB and introduces the additional constraint $3 \cdot x_1 + 2 \cdot x_2 = 19$ to define the restricted problem P(19). In order to strengthen P(19), (sharper) lower and upper bounds on the values of x_1 and x_2 are computed. This is achieved by solving the LP-relaxation of P(19) with the objectives of minimizing and maximizing x_1 and x_2 , respectively. This results in the two additional restrictions: $x_1 \in [2.06, 2.53]$ and $x_2 \in [5.69, 6.40]$.

Obviously, no integer value is feasible for x_1 within P(19) and, as a consequence, $LB = 19$ is no feasible objective function value of P. Thus, LB is increased to 20 and P(20) is constructed from P by adding the equation $3 \cdot x_1 + 2 \cdot x_2 = 20$. As new intervals of the variables we obtain $x_1 \in [2, 3.08]$ and $x_2 \in [5.38, 7]$. Due to the integrality of the variables, the upper bound on x_1 and the lower bound on x_2 can be set to 3 and 6, respectively. Adding those boundaries of the variables to P(20) and solving the LP-relaxation yields the solution $x^* = (2, 7)$ with $f(x^*) = 20$. This solution is integral and, therefore, optimal for P(20). Since $LB = 20$ is a lower bound for problem P, the solution is optimal for P, too, and the procedure stops. The additional constraints of P(19) and P(20) are shown in Figure 4.5. The bold parts of the lines indicate the feasible regions of the LP-relaxations.

Remark 4.7. Whenever the objective function has a large number of possible values one should prefer another search method (e.g. binary search) than successively considering values $LB, LB+1, \dots$. Starting with an interval $[LB, UB]$ of possible values, the mean element $M = \lfloor (LB+UB)/2 \rfloor$ is chosen and the restriction $f(x) \in [LB, M]$ is added to build P(M). If feasibility of P(M) can be precluded, then LB is set to $M+1$. Otherwise, UB is set to M. The search is repeated until $LB = UB$. If the objective function is real-valued, M must not be rounded and LB is set to M instead of $M+1$ in case of an infeasible restricted problem P(M). The search stops if the remaining interval is sufficiently small. For applications of binary search in the framework of destructive improvement, we refer to Heilmann and Schwindt (1997), Brucker and Knust (1998), and Schwindt (1998).

Remark 4.8. Sometimes, it is possible to utilize information drawn from the reduced P(LB) or the solution of its relaxation in order to increase LB by more than 1. As an example consider the critical sequence bound LBC3 from Section 4.1.1.1 which may be interpreted as some type of destructive improvement method. It starts with a critical path and restricts the schedule to the critical path

length LB. If it can identify another job j which can not be processed within its time window (completely), the lower bound can immediately be increased by $d_j - e_j^{\max}$. The other bounds of Section 4.1.1 are typical direct methods.

4.2.2 Applying Destructive Improvement to RCPSP

In the following, we describe how the destructive improvement technique can be adapted to RCPSP (cf. Klein and Scholl (1999)). First of all, different reduction techniques are introduced. Secondly, modifications of direct bound arguments which can benefit from the reduction in problem data are presented.

4.2.2.1 Reduction Techniques

In this section, some efficient reduction techniques for RCPSP are proposed. In the literature, reduction techniques are also referred to as immediate selection, edge finding, energetic reasoning, and interval consistency tests. A comprehensive survey on such techniques from a more theoretical point of view can be found in Dorndorf et al. (1999). Furthermore, techniques for constraint propagation originally designed for the job shop problem are discussed there (cf. also Blazewicz et al. (1996), Pesch and Tetzlaff (1996), and Nuijten and Le Pape (1998)). We renounce describing these techniques here due to the excessive number of publications in this field.

Reduction by Forward and Backward Pass. The most intuitive reduction technique has already been presented in Section 4.1.1.1. It consists of computing earliest and latest starting and finishing times by a modified forward and a backward pass using $LF_n = LBC_1$ to start the backward pass. The resulting time windows of jobs can be exploited by other reduction techniques and adapted bound arguments.

Reduction by Subprojects. This technique is based on identifying and evaluating subprojects of the original RCPSP instance. A subproject $SP(h,j)$ is defined by a pair of jobs (h,j) with j being a transitive successor of h and contains all jobs i which are successors of h and predecessors of j as well. Between h , j and the jobs of $SP(h,j)$, the original precedence constraints are preserved. The jobs h and j serve as dummy start and terminal nodes (with duration 0) of the subproject, respectively. If the optimal project completion time of $SP(h,j)$ is CT , the processing of job j starts at least CT periods after job h has been finished. Hence, CT or a lower bound on this value defines a *minimum finish-to-*

start time lag λ_{hj}^{FS} between h and j. Whenever λ_{hj}^{FS} exceeds the length of the critical path of at least one subproject SP(h,j), applying a forward and a backward pass to the overall project considering minimum time lags results in sharper time windows. Note that this may improve the bound by more than 1 as mentioned in Remark 4.8.

Minimum finish-to-start time lags may be derived by applying any (computationally inexpensive) bound to a subproject. Thus, the computational complexity of this reduction technique depends on the complexity of the bound arguments used and the number of subprojects being of order $O(n^2)$.

Example. In the example of Figure 4.1, p. 115, the pair (1,6) builds a subproject containing the jobs 3 and 4. The nodes 1 and 6 are the dummy start and terminal node, respectively. Since 3 and 4 can not be processed concurrently, a lower bound on the completion time of SP(1,6) is $d_3 + d_4 = 6$. Hence, due to $LS_6 = 5$, no feasible schedule can exist for $LB = 16$ (cf. Table 4.1, p. 115).

In general, any pair of jobs (h,j) with j being a transitive successor of h excluding (1,n) forms a subproject. But not all of these subprojects have to be evaluated. Consider a subproject SP(h,j) with job j having only one predecessor i. Since i does not compete for resources with other jobs of SP(h,j) and is a terminal job of a smaller subproject SP(h,i), the finish-to-start time lag between h and j can simply be computed by $\lambda_{hj}^{FS} = \lambda_{hi}^{FS} + d_i$. Analogously, the time lag λ_{hj}^{FS} of a subproject SP(h,j) with the start node having only a single successor i is obtained by $\lambda_{hj}^{FS} = d_i + \lambda_{ij}^{FS}$. Hence, only the subprojects SP(h,j) which have start and terminal nodes with at least two successors and two predecessors, respectively, have to be evaluated. For all pairs of jobs (h,j) with j being a direct successor of h, $\lambda_{hj}^{FS} = 0$.

When computing earliest and latest starting and finishing times using finish-to-start time lags, respectively, the set of all (direct and indirect) predecessors P_j^* and followers F_j^* of a job have to be considered. Then the following formulae have to be used in the modified forward and the backward pass, the latter being started with $LF_n = LB$ (cf. Section 3.2.1.1, pp. 77):

$$ES_j = \max\{EF_h + \lambda_{hj}^{FS} \mid h \in P_j^*\}; \quad (4.29)$$

$$LF_j = \min\{LS_h - \lambda_{jh}^{FS} \mid h \in F_j^*\}; \quad (4.30)$$

If $ES_n > LB$, a contradiction is present and LB can immediately be increased to ES_n .

Example. The following subprojects have to be evaluated for the project network of Figure 4.1, p. 115: SP(1,6), SP(1,7), SP(1,11), SP(3,11), SP(3,12), and SP(6,12). The finish-to-start time lags obtained for these subprojects by, e.g., computing LBC1, LBC2, LBC3, and LBN1 are listed in Table 4.4. Since a subsequent forward pass results in $ES_{12} = 19$, LB can be increased from 16 (= LBC1) to 19.

Table 4.4. Minimum time lags

SP(h,j)	SP(1,6)	SP(1,7)	SP(1,11)	SP(3,11)	SP(3,12)	SP(6,12)
λ_{hj}^{FS}	6	8	15	9	13	9

Reduction by Precedence. If the project completion time is restricted to LB some jobs must precede others to which they are not related by precedence. Following the logic of LBP1, two test projects are evaluated for each incompatible pair of jobs h and j by computing lower bounds $LB(h,j)$ and $LB(j,h)$ on the project durations. Going beyond the capabilities of LBP1, one of the arcs, w.l.o.g. say (h,j) , can temporarily be fixed whenever $LB(h,j) \leq LB < LB(j,h)$. Such a problem reduction is *valid* as long as $LB < LB(j,h)$. That is, additional arcs can be fixed using former reductions which are still valid for the current LB. If $LB < LB(h,j) \leq LB(j,h)$ is true for some incompatible pair currently considered, a contradiction occurs and LB can be increased. The new value is set to $LB(h,j)$ or the minimum project duration for which any of the arcs fixed earlier becomes invalid, whatever is smaller. Note that thereby bound increments larger than 1 are possible (cf. Remark 4.8).

By considering such additional precedence relations in the forward and the backward pass, sharper time windows for the jobs are obtained which may be exploited by other reduction techniques. Further precedence relations may be added considering incompatible triplets as in case of LBP2. Then, only such arcs (h,j) can be fixed for which any test project containing the reverse arc (j,h) gets a bound exceeding LB.

Reduction by precedence is an iterative process which is repeated as long as reductions are possible. Initially, all incompatible pairs and triplets are determined which takes $O(m \cdot n^3)$ time. Always after increasing LB or fixing an arc,

another iteration is performed which has complexity $O(n^3)$ due to checking all (predetermined) pairs and triplets again provided that LBC1 and LBC2 are used to evaluate the test projects.

Example. Examining the incompatible pair of jobs 7 and 8 gives $LB(7,8) = 19$ and $LB(8,7) = 20$. Hence, the arc $(7,8)$ can be fixed as long as $LB = 19$ is true. The feasibility of $LB = 19$ with arc $(7,8)$ is precluded by additionally considering the incompatible jobs 2 and 3 because of $LB(2,3) = 21$ and $LB(3,2) = 24$. As a consequence, the overall bound LB can be increased to 20 and the arc $(7,8)$ must be removed. Note that it is not possible to increase LB to 21, because the computation of this value, obtained for $TP(2,3)$, has been based on fixing the arc $(7,8)$, which is no longer valid.

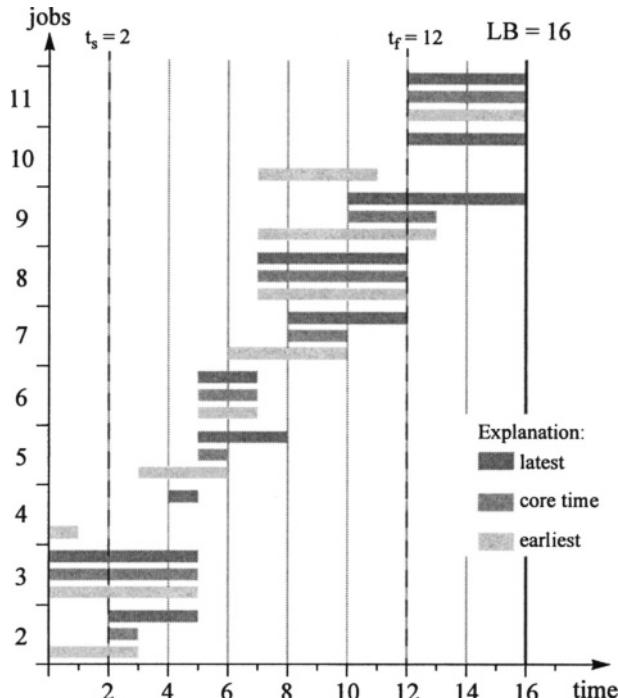


Figure 4.6. Gantt chart for $LB=16$

Reduction by Core Times. This reduction technique is based on the following idea: For a certain LB, earliest and latest starting and finishing times have been computed. Whenever the latest starting time LS_j of a job j is smaller than its earliest finishing time EF_j , job j must be active during $[LS_j, EF_j]$ as long as

LB is valid. This time interval is called *core time* of a job. In the context of resource leveling, jobs with a core time larger than 0 are also referred to as near-critical (cf. Neumann and Zimmermann (1997)).

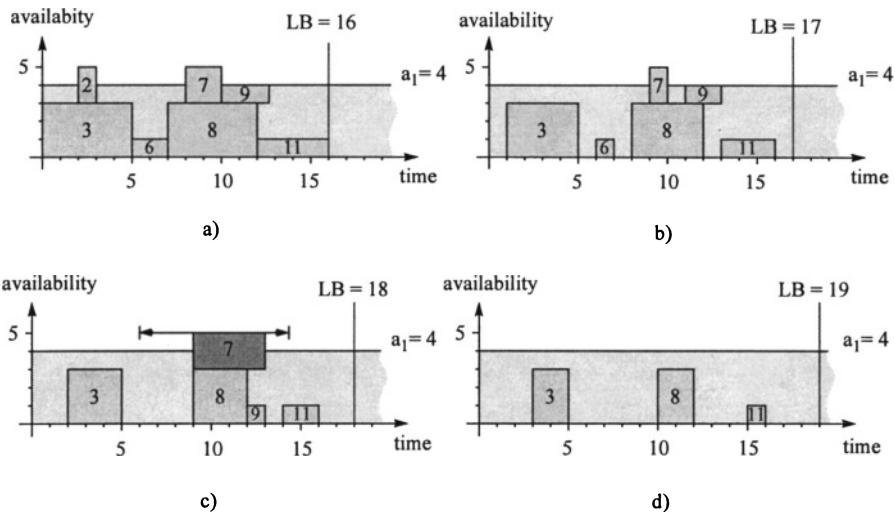
The reduction is started with scheduling all jobs (only) during their core times. If a resource conflict occurs in any period, LB can be increased. Otherwise, for each job we look for feasible time intervals during which it can be processed completely (in addition to the schedule built by the core times of the other jobs). In case only one according interval is available for a job j , the time window $[ES_j, LF_j]$ is restricted to this interval thereby also restricting time windows of predecessors and successors, respectively. This may result in new or enlarged core times. If no interval exists, LB can be increased.

Always after increasing LB or getting improved core times, another iteration of the reduction process is started. The time complexity $O(m \cdot n^2)$ of each iteration is due to searching possible time intervals for each of n jobs. The search requires checking $O(n)$ changes in the resource profile, which occur when core times start or end, and observing m resource types.

Example. The Gantt chart in Figure 4.6 shows each job, starting as early (light grey bar) and as late (dark grey bar) as possible (for $LB = LBC1 = 16$). The hatched bars in-between represent the core times of the jobs. For example, the core time of job 3, which lies on the critical path, is equal to its complete duration whereas job 2 has a core time of only a single period and jobs 4 and 10 have none.

Figure 4.7 contains capacity oriented Gantt charts for core times. Jobs are represented by rectangles with horizontal extents in length of their core times and vertical extents u_{j1} . Starting with the critical path length $LB = 16$ (cf. Figure 4.7 a), we recognize that in some periods the capacity requirements exceed the availability (periods 3, 9, 10) and LB is increased to 17.

For $LB = 17$, the latest starting times are increased by one reducing the core times accordingly (Figure 4.7 b). Since there still is a resource conflict in period 10, LB is increased again giving Figure 4.7 c) without a resource conflict caused by core times. Checking for feasible time intervals yields that processing job 7 within its time window $[6, 14]$ completely before or after job 8, which is incompatible to 7, is impossible (Figure 4.7 c). LB is increased to 19 resulting in Figure 4.7 d).

**Figure 4.7. Reduction by core times**

The only time interval during which job 7 can now be processed is [6,10]. Hence, its latest starting time is set to $LS_7 = 6$. Due to this reduction, the latest starting and finishing times of its predecessors can be decreased, too. With $LS_2 = 0$ and $LS_5 = 3$, the modified chart in Figure 4.8 is obtained. Due to the core times of jobs 2 and 7, job 3 can not be processed completely. For the resulting lower bound value $LB = 20$, no reduction is possible at all.

Applying Different Reduction Techniques. Usually, not only one reduction technique will be applied. Instead, it is more promising to use different techniques subsequently before trying to contradict the feasibility of the reduced problem instance. Therefore, it is necessary to put the general framework described in Section 4.2.1 (cf. Figure 4.4) into more concrete terms. The following steps are performed, after determining an initial bound value LB by direct methods:

Using the current LB as trial project duration, the reduction process is started by a forward and a backward pass using minimum time lags (see below). Afterwards, additional precedence constraints are introduced and core times are evaluated. This is continued until no further reduction is possible or a contradiction in the problem data occurs. In the first case, direct bound arguments are applied to the reduced instance (cf. Section 4.2.2.2). If feasibility of LB can be precluded, the bound value is increased and the reduction process is restarted maintaining only reductions which are still valid for the new LB .

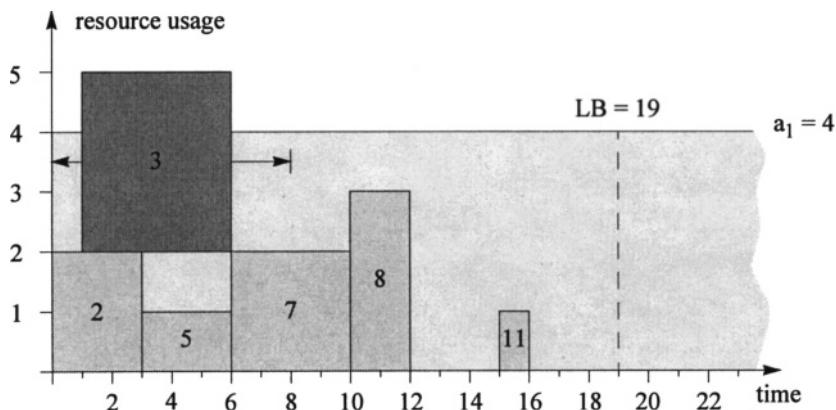


Figure 4.8. Core times for $LB = 19$

Reduction by subprojects is computationally more expensive than the other reduction concepts (cf. Section 7.3.2, pp. 278). Therefore, this technique is applied only once before starting the destructive improvement process to determine finish-to-start time lags between the jobs. All relevant subprojects are determined and sorted according to non-decreasing numbers of jobs contained. Examining them in this order ensures that a subproject is not considered earlier than all the ones it includes. That is, larger subprojects can benefit from the finish-to-start time lags between their jobs obtained by evaluating smaller subprojects. A finish-to-start time lag for any subproject $SP(h,j)$ is computed in the same manner as the lower bound for the complete project. That is, we start with applying direct bound arguments to the RCPSP instance defined by $SP(h,j)$ and perform the destructive improvement process as described above.

4.2.2.2 Lower Bound Arguments for Contradicting Feasibility

Within the destructive improvement approach, lower bound arguments are used in order to contradict the feasibility of a reduced problem when this is not possible during the reduction phase. Hence, not the value obtained is of primary interest but an answer to the question, whether a feasible schedule can exist or not. LBC1, LBC3, the parallel machine bounds LBM1 to LBM3 as well as the precedence bounds LBP1 and LPB2 can be applied directly to the reduced problem, profiting from the sharper time windows (heads and tails), whereas LBC2 and the node packing bounds must be strengthened in order to be successful in the destructive improvement context. Due to the rather poor quality

of the bound values obtained by the bin packing bounds, these are not applied within destructive improvement.

Capacity Bound LBC2. Consider a time interval $I = [t_s, t_f] \subset [0, LB]$ and the set S_I of jobs which have to be performed completely or partially within I given LB as maximal project duration. That is, S_I contains all jobs j with $EF_j > t_s$ and $LS_j < t_f$ which due to their time windows must be active within I for at least $d'_j = \min\{EF_j - t_s, t_f - LS_j, d_j\}$ periods. Computing LBC2 for this interval reveals whether or not it is not precluded to perform all jobs from S_I within an interval of length $t_f - t_s$:

$$LBC2(S_I) = \max \left\{ \left\lceil \sum_{j \in S_I} u_{jr} \cdot d'_j / a_r \right\rceil \mid r = 1, \dots, m \right\} \quad (4.31)$$

If $LBC2(S_I) > t_f - t_s$ for any S_I , the current bound value LB is contradicted and increased. In order not to examine all possible intervals, t_s and t_f are restricted to the earliest starting and latest finishing times of the jobs. Extending the application of destructive improvement to RCPSP with maximum time lags, Schwindt (1998 c, section 3.3) has defined sharper rules which time intervals have to be considered in order to detect all possible contradictions in the problem data.

With $LB = 16$, we have to consider the Gantt chart in Figure 4.6, p. 144. Examining $I = [2, 12]$, S_I contains the jobs 5, 6, 7, and 8 with their initial durations as well as the jobs 2, 3, and 9 with reduced durations $d'_2 = 1$, $d'_3 = 3$, and $d'_9 = 2$. Since $\lceil (3 \cdot 1 + 2 \cdot 1 + 4 \cdot 2 + 5 \cdot 3 + 1 \cdot 2 + 3 \cdot 3 + 2 \cdot 1) / 4 \rceil = \lceil 41 / 4 \rceil = 11$ exceeds the length of I , no feasible schedule can exist and LB is increased to 17.

Node Packing Bounds. Within the framework of destructive improvement, the notion of incompatibility may be extended. Besides precedence and resource incompatibility, two jobs are also incompatible to each other if concurrent processing is impossible due to their time windows. Hence, if the time windows of a job j just added to set S and a job h remaining in the list only overlap partially, the duration of h is reduced by the maximal number of periods the two jobs can be performed in parallel. For incompatible triplets, the same principle may be applied.

4.3 Lower Bound Methods for GRCPSP

In this section, we describe how the simple lower bound arguments as well as the destructive improvement method introduced in the two sections before can be adapted for GRCPSP (cf. Section 3.2.2, pp. 89). First of all, the application of the simple lower bound arguments is shown in Section 4.3.1. Subsequently, Section 4.3.2 is devoted to the presentation of destructive improvement. To ease the presentation, a new example is introduced in the Figures 4.9 and 4.10. The example is kept as simple as possible, e.g., we renounce considering release and due dates though these are integrated in the computation of the bound arguments. Furthermore, the descriptions are restricted to giving the basic idea of the respective bound arguments.

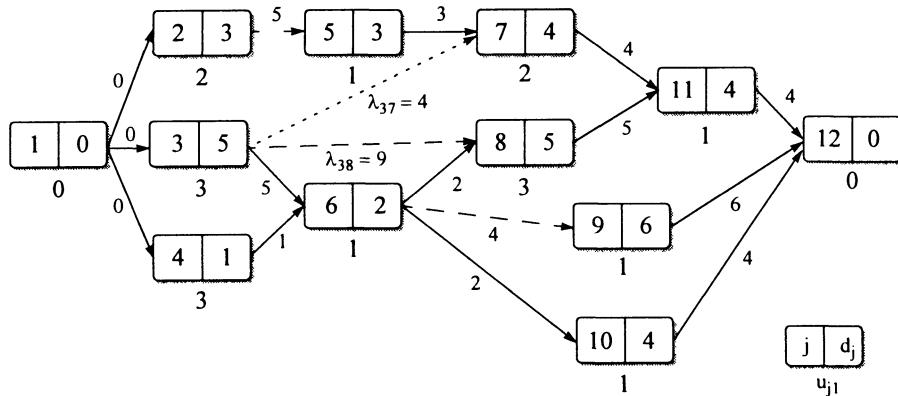


Figure 4.9. Start-to-start project network

Figure 4.9 shows a start-to-start project network with $n = 12$ jobs, which has been derived from the RCPSP example used in the previous sections. In addition to the job durations and usages of the single resource type, the minimum start-to-start time lags λ_{ij} are denoted below each arc (i,j) . A dotted arc indicates that the execution of a pair of jobs related by precedence can partially overlap. This is, e.g., true for the jobs 3 and 7. Due to $d_3 = 5$ and $\lambda_{37} = 4$, job 7 can already be started when job 3 has still to be processed for another period. Dashed arcs depict precedence relationships which require a successor job to wait for a number of periods after completing the predecessor one. For example, at least four periods have to pass away between terminating job 3 and starting job 8. Note that the arc $(3,8)$ is not redundant though a path between these

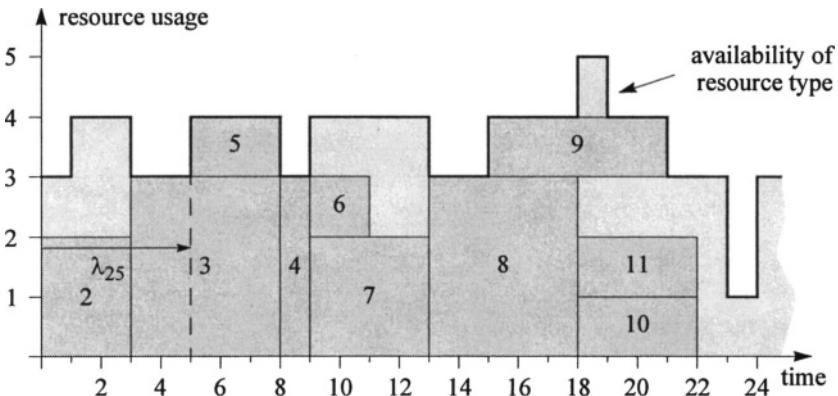


Figure 4.10. Optimal solution

two jobs containing job 6 exists. However, this path has only a length of $\lambda_{36} + \lambda_{68} = 7$ which is smaller than $\lambda_{38} = 9$.

An *availability profile* for the single resource type considered is defined in the capacity-oriented Gantt chart of Figure 4.10 for the periods $t = 1, \dots, 24$. The overall planning horizon is assumed to be $\bar{T} = 28$ with $a_{1t} = 3$ for the periods $t = 25, \dots, 28$. Hence, the minimum and maximum availability of resource type 1 is 1 and 5 capacity units, respectively.

Furthermore, an optimal solution for the so-defined GRCPSP instance with a project completion time of $CT = 22$ is given. At the beginning of period 1, job 2 is started. Due to the resource constraints, neither job 3 nor job 4 can be processed concurrently. After finishing job 2, only the jobs 3 and 4 can be put into progress among which the first one is chosen. Job 5 is not eligible yet due to the minimum time lag $\lambda_{25} = 5$. Therefore, it is executed in parallel to job 3 in the periods 5, 6, and 7 where the resource availability increases from 3 to 4 units. Subsequently, job 4 is performed. Now, the jobs 6 and 7 are immediately scheduled. Job 8 can not begin earlier than in period 13 without violating the resource constraints. Finally, the jobs 9, 10, and 11 are processed.

4.3.1 Simple Bound Arguments

In the following, the application of the simple bound arguments presented for RCPSP in Section 4.1.1 to GRCPSP is explained. The major differences of GRCPSP in comparison to RCPSP concern the existence of start-to-start prece-

dence relationships with minimum time lags and fluctuations in the resource availabilities. Due to these differences, the following difficulties arise in adapting the lower bound arguments:

- Some of the bound arguments as well as their extensions have to be omitted in order not to yield incorrect bound values. For example, this is the case for the critical sequence bound LBC3 and the extended node packing bound LBN2 (cf. Sections 4.3.1.1 and 4.3.1.2 for corresponding discussions).
- For other bound arguments, like the bin packing bounds, the adaptation is not worthwhile because their computation requires a constant availability for each resource type r considered. In general, this requirement can be fulfilled by assuming a constant availability of $a_r^{\max} = \max\{a_{rt} \mid t = 1, \dots, \bar{T}\}$ for each resource type $r = 1, \dots, m$. However, using this "work-around", results in these arguments obtaining rather weak lower bound values. In our example, we yield $a_r^{\max} = 5$ though this capacity supply is only provided in a single period $t = 19$.
- Finally, some bound arguments rely on definitions which have to be refined for GRCPSP. In particular, this is true concerning the incompatibility of jobs. In case of RCPSP, a pair or triplet of jobs is said to be incompatible, if it is related by precedence or its total resource demand exceeds the per period availability for any resource type. However, this definition can not immediately be transferred to GRCPSP due to the following reasons. First of all, even the execution of two jobs h and j connected by a (transitive) precedence relationship may partially overlap when the length of the longest path λ_{hj}^* connecting them is smaller than the duration d_h of job h , i.e., $\lambda_{hj}^* < d_h$ (cf. Definition 3.4, p. 75). Secondly, the parallel execution of a pair or triplet of jobs may not violate the resource constraints in some periods whereas it causes a resource conflict in others.

The most simple form of overcoming the latter difficulty again consists in referring to the values a_r^{\max} determined for each resource type r . Then, e.g., the resource incompatibility of a pair (h,j) can be verified by checking whether $u_{hr} + u_{jr} > a_r^{\max}$ for any resource type r . However, as stated above, this approach can be rather ineffective. In our example, only the job pair $(3,4)$ would be considered to be incompatible. Therefore, in order to yield strong lower bound values, we also have to examine *partially incompatible* pairs and triplets of jobs which can be processed in parallel only for a number of periods smaller than their respective durations (cf. Section 4.3.1.2).

4.3.1.1 Critical Path and Capacity Bounds

Both, the critical path bound LBC1 and the capacity bound LBC2 have to be modified to work for GRCPSP. The critical sequence bound LBC3 can not be used for GRCPSP at all due to resulting in incorrect bound values. For an according example we refer to Demeulemeester (1992, section 3.2.4).

Critical Path Bound (LBC1'). According to RCPSP, the resource constraints can be discarded. Subsequently, the length of a critical path can be computed in $O(n^2)$ time giving a lower bound on the project completion time. A description of the corresponding forward and backward pass considering the start-to-start project network as well as release and due dates is contained in Section 3.2.2.1, pp. 91. The earliest starting and finishing times which are obtained for the example of Figure 4.9 are given in Table 4.5, p. 153. The critical path contains the jobs $CP = \langle 1, 3, 8, 11, 12 \rangle$, and we yield $LBC1' = 18$.

Capacity Bound (LBC2'). Only a single resource type r is considered for which the total capacity requirement $\sum_{j=1}^n u_{jr} \cdot d_j$ of all jobs is determined. This capacity requirement is compared to the capacity available in the periods $1, 2, \dots$. That is, we compute the earliest period $LBC2'(r)$ at the end of which the cumulated capacity supply is at least as large as the total requirement:

$$LBC2'(r) = \min \left\{ t \mid t \in [1, \dots, \bar{T}] \text{ and } \sum_{j=1}^n u_{jr} \cdot d_j \leq \sum_{q=1}^t a_{rq} \right\} \quad (4.32)$$

The maximum bound $LBC2'(r)$ yielded for all resource types r defines a lower bound value $LBC2'$. In the example, the total demand of resource type 1 is 66 capacity units. This amount has been provided until the end of period 18 such that $LBC2' = 18$. Due to computing the total resource demand of all n jobs and examining the availability of each resource type in each period $t = 1, \dots, \bar{T}$, the computational effort required for computing $LBC2'$ is of the order $O(m \cdot (n + \bar{T}))$.

Remark 4.9. Within an according procedure for computing $LBC2'$, the actual computational effort may be considerably reduced by exploiting that fluctuations in the resource availability profiles may only occur at the end of certain periods. Let $t^s < t^f$ denote two time points between which the availability of a

resource type r does not change. Then, the total resource availability between these time points can simply be computed by $a_{r,t^s+1} \cdot (t^f - t^s)$. However, in the worst case the availability may change at the end of each period of the planning horizon.

4.3.1.2 Node Packing Bounds

The node packing bounds presented for RCPSP rely on identifying incompatible pairs and triplets of jobs. As stated above, when adapting these bound arguments for GRCPSP, we also have to consider pairs and triplets of jobs which are only *partially incompatible*, i.e., which can be processed in parallel only for a number of periods smaller than their respective durations.

Node Packing Bound (LBN1'). In order to consider partially incompatible pairs, the version of LBN1 used within the destructive improvement framework is applied. For RCPSP, it does not delete a job h compatible to a job j just added to S completely from the list. Instead, it reduces the duration of job h by the maximum number of periods Δ_{hj} in which the two jobs may be processed in parallel. In the following, we explain the computation of the values Δ_{hj} for a given GRCPSP instance. Note that due to $\Delta_{hj} = \Delta_{jh}$ the values Δ_{hj} have to be computed only for pairs (h,j) with $h < j$. Having these values on hand, LBN1' can be calculated as described in the Sections 4.1.1.3 and 4.2.2.2. In order to apply the same sorting criterion as described for RCPSP, two jobs h and j are considered to be (completely) incompatible, if $\Delta_{hj} = 0$ has been obtained.

Table 4.5. Time windows for $LF_n = LS_n = 28$

j	1	2	3	4	5	6	7	8	9	10	11	12
ES_j	0	0	0	0	5	5	8	9	9	7	14	18
EF_j	0	3	5	1	8	7	12	14	15	11	18	18
LS_j	10	12	10	16	17	17	20	19	22	24	24	28
LF_j	10	15	15	17	20	19	24	24	28	28	28	28

First of all, *time windows* considering the release and due dates of jobs are computed by a forward and a backward pass as described in Section 3.2.2.1, pp. 91, respectively. The backward pass is initialized with an upper bound on the project completion time which can either correspond to the planning horizon \bar{T}

or to an upper bound UB which has been determined by applying one of the heuristics described in chapter 5. The time windows obtained for the example project starting the backward pass with $LF_n = LS_n = \bar{T} = 28$ are given in Table 4.5.

Now, the earliest (starting) time point t_{hj}^s from which two jobs h and j with $h < j$ can be executed in parallel can be computed by $t_{hj}^s = \max\{ES_h, ES_j\}$. Accordingly, the time point t_{hj}^f at which concurrent processing finishes the latest is determined by $t_{hj}^f = \min\{LF_h, LF_j\}$. In case of $t_{hj}^f \leq t_{hj}^s$, the execution of the jobs can not overlap at all. Otherwise, they may be performed in parallel for at most $t_{hj}^f - t_{hj}^s$ periods. However, the actual number may be smaller because the total resource demand of the two jobs may exceed the availability of a resource type in one or more of the periods $t = t_{hj}^s + 1, \dots, t_{hj}^f$ or a (transitive) precedence relationship with $\lambda_{hj}^* > 0$ may exist between h and j :

- In the first case, all sub-intervals of $[t_{hj}^s, t_{hj}^f]$ in which the jobs can be performed simultaneously without causing a resource conflict have to be determined. Since preemption of jobs is not allowed, the length of the longest of these sub-intervals defines an upper bound on the number of periods Δ_{hj}^{ra} during which parallel processing of the jobs h and j is possible. The superscript "ra" indicates that this upper bound is due to the restricted resource availabilities. In our example, we yield $t_{78}^s = 9$ and $t_{78}^f = 24$ for the jobs 7 and 8 among which no precedence relationship exists. Only in period 19, the availability $a_{19} = 5$ of the single resource type equals their total demand $a_{71} + a_{81} = 5$. Hence, $\Delta_{78}^{ra} = 1$ is calculated. For the jobs 9 and 11, we obtain $t_{9,11}^s = 14$ and $t_{9,11}^f = 28$. However, the jobs can not be processed concurrently during the complete interval $[14, 28]$ because of $a_{1,24} = 1$. That is, the sub-intervals $[14, 23]$ and $[24, 28]$ are determined. With the first one having a length of 9, we yield $\Delta_{9,11}^{ra} = 9$.
- In the second case, if job h is a (transitive) predecessor of job j , the maximum number of periods for which the execution of the jobs can overlap due to the precedence constraints is limited by $\Delta_{hj}^{pr} = \max\{0, d_h - \lambda_{hj}^*\}$ which is denoted by the according superscript "pr". Otherwise, $\Delta_{hj}^{pr} = \infty$. In our example, we obtain $\Delta_{37}^{pr} = \max\{0, 5 - 4\} = 1$ for the jobs 3 and 7.

Finally, Δ_{hj} can be computed by the following formula:

$$\Delta_{hj} = \min \left\{ d_h, d_j, \Delta_{hj}^{ra}, \Delta_{hj}^{pr} \right\} \quad (4.33)$$

Remark 4.10. Note that during the application of LBN1' the initial durations d_h and d_j of the jobs h and j may have been reduced due to (partially) compatible jobs which have already been entered into S . In this case, Δ_{hj}^r has to be updated accordingly. By the way of contrast, the values of Δ_{hj}^{ra} and Δ_{hj}^{pr} do not change.

Example. The values Δ_{hj} obtained for our example project with $\bar{T} = 28$ are given in Table 4.6. Recalling the sorting criterion used for LBN1' (cf. Section 4.1.1.3, pp. 120) and considering job pairs (h,j) with $\Delta_{hj} = 0$ to be incompatible, we yield the list $L = \langle 3, 8, 11, 4, 2, 6, 7, 5, 9, 10 \rangle$ due to the jobs 3, 8, and 11 being on the critical path. The jobs 3, 8, 11, and 4 are successively included into the set S because they can not be processed in parallel in any period of the planning horizon. This results in removing the jobs 5, 9, and 10 from the list completely. Furthermore, the duration of job 7 is reduced to $d_7 = 4 - \Delta_{78} = 3$. Subsequently entering job 2 into S results in deleting job 6 from the list. Finally, only job 7 remains in the list and, hence, is added to S with its reduced duration of $d_7 = 3$ such that we obtain a lower bound value of $LBN1' = 5 + 5 + 4 + 1 + 3 + 3 = 21$.

Table 4.6. Values of Δ_{hi} for example project

The values λ_{hj}^* for all jobs $(h,j) \in A^*$ can be computed in $O(n^3)$ time (cf., e.g., Floyd (1962), Dantzig (1967), and Domschke (1995, section 5.3)). Calculating earliest and latest starting and finishing times and, hence, the values t_{hj}^s and t_{hj}^f is of the order $O(n^2)$. Finally, determining the values Δ_{hj}^{ra} takes $O(m \cdot n^2 \cdot \bar{T})$ time. With these data available, the computational complexity of evaluating the list once, i.e., without rotation, is $O(n^2)$. Therefore, the total effort of computing LBC2' without rotation is $O(m \cdot n^2 \cdot \bar{T} + n^3)$.

Extended Node Packing Bound (LBN2'). For RCPSP, LBN1 is extended to LBN2 by applying the arguments LBC1 and LBC2 to the residual project instances defined by the jobs remaining in the list at the end of each iteration. This is not feasible for GRCPSP and LBC1' due to the fact that the computation of the critical path no longer depends on the duration of jobs but on minimum time lags between jobs which can not be adjusted appropriately. In case of LBC2', another problem arises. Computing the number of periods required for providing a resource availability exceeding the total demand of the jobs remaining in the current list is difficult. This is due to the circumstance that these jobs may be performed in arbitrary periods of the planning horizon. Therefore, formula (4.32) can not be applied as it is, because it relies on the assumption that the execution of the jobs considered starts in period 1. Again, this problem can be avoided by assuming a per period availability a_r^{\max} for each resource type $r = 1, \dots, m$ with the disadvantage of possibly resulting in rather weak bound values. Therefore, adapting LBN2 to GRCPSP is not worthwhile.

Generalized Node Packing Bound (LBN3'). This bound argument considers incompatible triplets of jobs. According to incompatible pairs, the problem for GRCPSP consists in determining such triplets. To overcome this problem, the same approach as for LBN1' may be realized. That is, the extended version described in Remark 4.2 may be applied in combination with the modified concept of partial incompatibility used in the destructive improvement framework (cf. Section 4.2.2). For this purpose, a value Δ_{hij} is computed for each triplet (h,i,j) (with $h < i < j$) of jobs denoting the number of periods for which their execution can overlap at most. Since, basically, this value can be calculated according to Δ_{hj} in case of pairs (h,j) , we renounce giving a detailed description. Furthermore, following the argumentation for LBN2', no lower bound arguments are applied to the residual instance defined by the jobs still in the list L after each iteration. Due to considering triplets of jobs, the time required for

computing LBN3' is of the order $O(m \cdot n^3 \cdot \bar{T})$ when rotation of the list L is omitted.

4.3.1.3 Parallel Machine Bounds

In Section 4.1.1.4, the parallel machine bounds LBM1, LBM2, and LBM3 are introduced for RCPSP. According to the bin packing bounds, LBM1 is based on discarding all but one resource type. For this resource type r , a set S of jobs from which at most k can be processed in parallel without causing a resource conflict is determined. A job j is included in S if its resource usage u_{jr} is not less than $\lfloor a_r / (k+1) \rfloor + 1$. In order to apply this simple decision rule for GRCPSP, a per period availability a_r^{\max} for each resource type $r = 1, \dots, m$ has to be assumed. As pointed out before, this may result in the bound argument possibly yielding rather weak values. Therefore, we omit giving a description of an adapted version of LBM1. By the way of contrast, the bound arguments LBM2 and LBM3 are based on the incompatibility of pairs and triplets of jobs, respectively. Hence, they may be applied to GRCPSP following the logic of LBN1' and LBN3'.

One-Machine Bound (LBM2'). For computing LBM2', heads and tails have to be defined for each job $j \in J$. According to the RCPSP version, this requires performing a forward and a backward pass, respectively. Then, for each job j , we yield $\alpha_j = ES_j$ and $\omega_j = LF_n - LF_j$. Furthermore, the values Δ_{hj} have to be calculated as described above. Subsequently, a set S may be determined heuristically using the sorting criterion of LBM2 (cf. Section 4.1.1.4). Each time having entered a job h into S , the duration of each job j with $\Delta_{hj} > 0$ remaining in the list is modified and formula (4.5), p. 125, is evaluated. Like with LBN1', the effort of computing LBM2' starting with a single job q is of the order $O(m \cdot n^2 \cdot \bar{T} + n^3)$.

Table 4.7. Heads and tails for $LF_n = LS_n = 28$

j	1	2	3	4	5	6	7	8	9	10	11	12
α_j	0	0	0	0	5	5	8	9	9	7	14	18
ω_j	18	13	13	11	8	9	4	4	0	0	0	0

Example. The modified one-machine bound is computed starting with the job $q = 3$. Calculating heads and tails for our example results in the values given in

Table 4.7. By applying the sorting criterion described for LBM2 (cf. Section 4.1.1.4), we obtain the list $L = \langle 3, 2, 4, 6, 5, 7, 8, 10, 9, 11 \rangle$ excluding the dummy jobs. At the beginning, the jobs 3 and 2 are entered into S . As a consequence, the jobs 6 and 5 are eliminated from the list whereas the durations of jobs 9 and 10 are reduced to $d_9 = 3$ and $d_{10} = 1$, respectively. Since job 3 and job 2 have identical heads and tails of $\alpha_j = 0$ and $\omega_j = 13$, evaluating formula (4.5) yields the lower bound value $M(\{3, 2\}) = 0 + 8 + 13 = 21$. This value is achieved again for $S = \{3, 2, 4, 7, 8\}$. After adding job 7 to S , the remainders of job 9 and 10 are discarded from the list and the duration of job 8 is decreased by one period due to $\Delta_{78} = 1$. Hence, $M(\{3, 2, 4, 7, 8\}) = 0 + (5 + 3 + 1 + 4 + 4) + 4 = 21$ is yielded.

Two-Machine Bound (LBM3'). This bound argument extends LBM2' by considering incompatible triplets of jobs. Following the logic of LBN3', values Δ_{hij} have to be computed for each triplet (h, i, j) of jobs with $(h < i < j)$ which requires $O(m \cdot n^3 \cdot \bar{T})$ time. Subsequently, LBM3' can be computed as described in Section 4.1.1.4.

4.3.1.4 Precedence Based Bounds

According to the bound arguments described before, the precedence based bounds are based on examining incompatible pairs and triplets of jobs. Again, for effectively applying them to GRCPSP, the maximum number of periods Δ_{hj} and Δ_{hij} for which each pair (h, j) or triplet (h, i, j) of jobs can be processed in parallel without violating resource and precedence constraints have to be computed, respectively.

Precedence Bound 1 (LB1'). This bound considers pairs (h, j) with $h < j$ which are not related by precedence. If $\min\{d_h, d_j\} > \Delta_{hj}$, those jobs must not be processed in parallel for at least $\min\{d_h, d_j\} - \Delta_{hj}$ periods such that a disjunctive precedence relationship may be defined among them. The first test project $TP(h, j)$ is obtained by introducing an arc (h, j) with a minimum time lag of $\lambda_{hj} = \min\{d_h, d_j\} - \Delta_{hj}$. The second one $TP(j, h)$ is yielded accordingly by defining an arc (j, h) with $\lambda_{jh} = \min\{d_h, d_j\} - \Delta_{hj}$. Having evaluated both test projects by calculating respective bound values $LB(h, j)$ and $LB(j, h)$, the smaller one defines a lower bound value for the original GRCPSP instance. The lower bound LB1' is then obtained by performing this test to all relevant pairs and taking the maximum value found. If the evaluation of a test project $TP(h, j)$

is restricted to the application of LBC1', this can be done efficiently by applying the formula $\alpha_h + d_h + d_j - \Delta_{hj} + \omega_j$ (cf. Remark 4.4, p. 129).

Example. Consider the job pair (7, 8). For the test project TP(7, 8), the application of LBC1' results in $LB(7, 8) = 8 + 4 + 5 - 1 + 4 = 20$ while for TP(8, 7) the value $LB(8, 7) = 9 + 5 + 4 - 1 + 4 = 21$ is obtained. Only evaluating the pair (2, 3) yields a better bound value of $LBP1' = 21$. Both jobs possess identical heads and tails of $\alpha_j = 0$ and $\omega_j = 13$ and can not be executed in parallel ($\Delta_{hj} = 0$) such that $LBP1' = LB(2, 3) = LB(3, 2) = 5 + 3 + 13 = 21$.

Precedence Bound 2 (LBP2'). This bound argument extends LBP1' by examining triplets of jobs (h, i, j) among which no precedence relationships exist. Following the above argumentation for LBP1', these jobs must not be executed concurrently for at least $\max\{0, \min\{d_h, d_i, d_j\} - \Delta_{hij}\}$ periods. As a result, different test projects can be derived for this triplet as this is done for RCPSP. Since this process is basically described for LBP1' above, we renounce giving a description. Subsequently, these test projects can be evaluated by applying LBC1'. The smallest value computed for any of them represents a lower bound for the original GRCPSP instance. Examining all such triplets of jobs accordingly and taking the maximum bound value obtained yields a lower bound LBP2'.

4.3.2 Destructive Improvement

When applying destructive improvement to GRCPSP, both the reduction techniques as well as the bound arguments for contradicting feasibility have to be modified. With these modifications destructive improvement can be employed as presented for RCPSP in Section 4.2.2.

Reduction Techniques. In Section 4.2.2.1, four different reduction techniques are introduced for RCPSP. In the following, we briefly describe the adaptation of these techniques to GRCPSP:

- *Reduction by Forward and Backward Pass.* This intuitive reduction technique has already been used for computing time windows to enhance the node packing and parallel machine bounds. Within destructive improvement, the backward pass has to be initialized with $LF_n = LS_n = LB$ with LB denoting the trial lower bound value currently being examined.

- *Reduction by Subprojects.* Adapting this technique to GRCPSP is difficult for two reasons. For a subproject (h,j) , the execution of the start job h and the end job j of a subproject may overlap with the processing of other jobs contained within due to considering minimum time lags. Therefore, those jobs must be considered only partially when computing a respective lower bound value. Secondly, some bound arguments like, e.g., LBC2' can not be used at all, because the actual starting time of the start job h and, hence, of the subproject can not be determined a priori. More figuratively speaking, the temporal position of the subproject in the resource availability profile(s) can not be defined. Furthermore, the efficiency of other bound arguments also depends on this value. By assuming a starting time of ES_h determined by a forward pass, information yielded by evaluating other subprojects may not be utilized efficiently. Additionally taking into account that this technique is the computationally most expensive one, its adaptation to GRCPSP does not seem to be worthwhile.
- *Reduction by Precedence.* For RCPSP, this technique is based on applying the logic of the precedence bounds LBP1 and LBP2. In order to exploit this technique also for GRCPSP, the modified bound arguments LBP1' and LBP2' have to be used.
- *Reduction by Core Times.* The core time of a job j is defined by the interval $[LS_j, EF_j]$. These values are available after performing a reduction by a forward and a backward pass. Subsequently, a reduction by core times can be performed as described in Section 4.2.2.1, pp. 141. Note that already for RCPSP fluctuations in the resource availability have to be considered due to the usage of jobs already scheduled during their core times.

Lower Bound Arguments for Contradicting Feasibility. All lower bound arguments which have been described in the previous section may be applied for contradicting feasibility of a reduced problem. LBC2' is strengthened by examining different time intervals according to LBC2 for RCPSP. For each time interval considered, the total availability of each resource type is compared to the demand of the jobs which have (partially) to be processed within. The computation of the node packing and the parallel machine bounds requires determining the maximum number of periods Δ_{hj} and Δ_{hij} for which each pair (h,j) and triplet (h,i,j) of jobs can be executed concurrently. These values may be recomputed each time no further reductions are possible.

5 Heuristic Procedures

Within the project planning process, the scheduling of jobs necessary for successfully completing a project as early as possible is a major task which becomes challenging and mathematically complex as soon as resource constraints are explicitly considered (cf. Chapter 3). However, when a software is used for accomplishing this task, project managers expect to get proposals for favorable and realizable schedules within short time. So, fast but powerful methods are required for finding such schedules.

Taking the computational complexity of resource-constrained project scheduling problems into account, this requirement on computer based scheduling methods, i.e., being fast and effective, can only be met by applying heuristics. Most of the *heuristics* presented for solving RCPSP either belong to the class of constructive methods or the class of improvement methods. *Constructive methods* start from an initial state and perform a set of operations leading to a (feasible) solution, e.g., by selecting and adding solution components step by step. In the context of resource-constrained project scheduling, constructive methods most commonly commence with none of the project's jobs being scheduled. Subsequently, a schedule is constructed by selecting a subset of jobs in each step and assigning feasible starting times to these jobs until all jobs have been considered. For this purpose, the jobs are usually ranked using priority rules. Therefore, we give preference to the more common term *priority-rule based heuristics* in the following. According to heuristics as well as possible extensions like multi-pass heuristics, are discussed for RCPSP and GRCPSP in Section 5.2. In order to ease their presentation, different types of schedules are previously introduced in Section 5.1. Note that due to GRCPSP being NP-hard in the strong sense, for some instances no feasible solution may exist at all or, though there is one, heuristics might fail in determining it (cf. Section 7.4.1).

Improvement methods are applied to improve on an initial solution which has been determined by some constructive method. This is done by successively executing operations which transform one solution into another. Traditional

methods of this type only permit operations leading to a solution improving the best known objective function value in each step. More evolved methods avoid getting stuck in a local optimum by also allowing operations resulting in an intermediate deterioration of the objective function value. In order to prevent such approaches from considering the same set of solutions repeatedly, *meta-heuristics* such as simulated annealing and tabu search have been introduced. A further class of meta-heuristics is represented by *evolutionary algorithms* among which the genetic algorithms are most popular. In Section 5.3, meta-heuristic based approaches designed for solving RCPSP are presented. Furthermore, a tabu search procedure for GRCPSP is proposed.

Further heuristic methods developed for RCPSP, which are not discussed in detail here, are disjunctive arc based heuristics (cf. Shaffer et al. (1965), Alvarez-Valdés and Tamarit (1989 b), and Bell and Han (1991)), truncated branch and bound (cf. Pollack-Johnson (1995), Sprecher (1997), and Section 7.5.2.3, pp. 321) as well as integer programming based heuristics (cf. Oguz and Bala (1994)). Furthermore, see Yang et al. (1989, 1993 b), and Zamani and Shue (1998). For recent surveys on heuristic algorithms for solving RCPSP, we refer to Kolisch (1995, chapter 5), Hartmann and Kolisch (1998), and Kolisch and Hartmann (1999).

5.1 Types of Schedules

In the sequel, a number of terms and definitions concerning different types of schedules are introduced which ease the presentation of heuristic and exact procedures. Note that the definitions are valid for both, RCPSP and GRCPSP such that an explicit distinction is omitted. For the purpose of illustration, the simple GRCPSP example defined by the start-to-start project network of Figure 5.1 and the availability profile of Figure 5.2 is introduced.

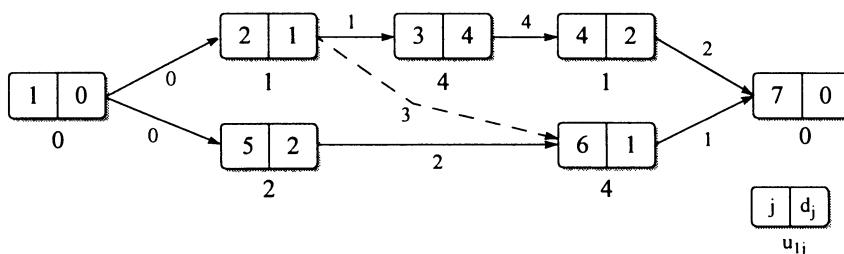


Figure 5.1. Example for different types of schedules

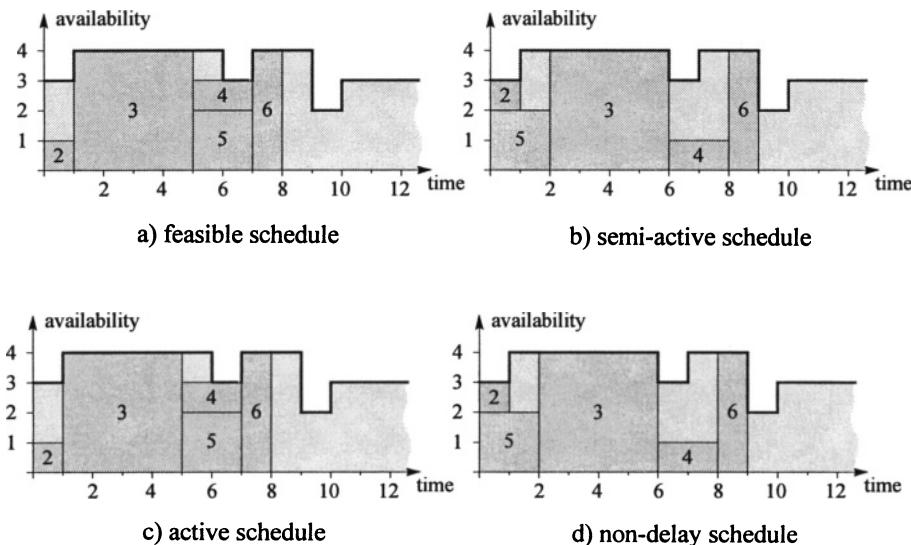


Figure 5.2. Types of schedules

As stated before, RCPSP and GRCPSP consist of determining a schedule such that the project is terminated as early as possible without violating the precedence and resource constraints. To be more specific, we introduce the following definitions. Table 5.1 provides a summary of the most important notations.

Definition 5.1. A (complete) *schedule* CS assigns a scheduled starting (finishing) time $SS_j(CS)$ ($SF_j(CS) := SS_j(CS) + d_j$) to each job $j = 1, \dots, n$, i.e., job j is performed in the periods $SS_j + 1(CS), \dots, SF_j(CS)$. CS is called *feasible* if the precedence constraints as well as the resource constraints are observed.

Remark 5.1. In order to enhance readability, we prefer the notation SS_j and SF_j to the one including "(CS)" when the context reveals which (partial) schedule is considered (see below).

Example. Figure 5.2 a) shows a feasible schedule for the example problem. The scheduled starting (finishing) time of job $j = 3$ is $SS_3 = 1$ ($SF_3 = 5$). Hence, job 3 is active in the periods $t \in [2, \dots, 5]$.

While constructing a schedule, not all jobs may have been assigned a starting time yet. That is, within solution procedures also partial schedules have to be considered.

Table 5.1. Notations and Definitions

Notation	Definition
CS	feasible (complete) schedule
PS	feasible partial schedule
$J(PS)$	jobs which are scheduled in a partial schedule PS
$SS_j(PS)$	scheduled starting time for job j in PS; $j \in J(PS)$
$SF_j(PS)$	scheduled finishing time for job j in PS; $j \in J(PS)$
$ES_j(PS)$	schedule dependent earliest starting time for job j ; $j \in J - J(PS)$
$EF_j(PS)$	schedule dependent earliest finishing time for job j ; $j \in J - J(PS)$
$A(PS)$	jobs available for the partial schedule PS; (= { j $j \in J - J(PS)$ and $P_j \subseteq J(PS)$ })
$E(PS,t)$	jobs eligible for the partial schedule PS at time point t ; (= { j $j \in A(PS)$ and $ES_j(PS) \leq t$ })

Definition 5.2. In a (feasible) *partial schedule* PS, starting times are only fixed for a subset $J(PS)$ of jobs. Partial schedules may successively be completed to schedules by assigning starting times to *available* jobs $j \in J - J(PS)$, i.e., jobs for which all predecessors are already contained in $J(PS)$. The set of all available jobs is denoted by $A(PS)$.

Depending on the jobs scheduled so far, the earliest starting and finishing times of unscheduled jobs might be increased in comparison to the values obtained by the critical path analysis.

Definition 5.3. For each partial schedule PS, a *modified forward pass* may be performed yielding partial schedule dependent earliest starting and finishing times $ES_j(PS)$ and $EF_j(PS)$ for all jobs $j \in J - J(PS)$. For this purpose, we simply set $ES_j(PS) = SS_j$ and $EF_j(PS) = SF_j$ for all jobs $j \in J(PS)$. Subsequently, a standard forward pass can be applied for all unscheduled jobs. An available job $j \in A(PS)$ is said to be *eligible* at a time point t , if $ES_j(PS) \leq t$. The jobs eligible at a time point t form the set $E(PS, t)$.

Furthermore, the scheduled jobs $j \in J(PS)$ absorb resources in those periods in which they are processed decreasing the capacity available for the execution of the unscheduled jobs.

Definition 5.4. A job j is said to be *active* in a period t , if it starts at the beginning of period t or earlier ($SS_j \leq t - 1$) and does not finish before the end of

period t ($SF_j \geq t$). For each resource type $r = 1, \dots, m$, the *residual availability* remaining in a period t can be determined by computing the difference between the per period availability $a_{r(t)}$ and the total resource demand of the jobs being active in t .

Example. Consider the schedule given in Figure 5.2 d). For the partial schedule PS which is defined by $J(PS) = \{2, 5\}$, the jobs 3 and 6 are available, i.e., $A(PS) = \{3, 6\}$. However, at the time point $t = 2$ when job 2 and job 5 are terminated, only job 3 is eligible, because the execution of job 6 can not be started before $t = 3$ due to the minimum time lag $\lambda_{26} = 3$. In period 2, only job 5 is active with a resource demand of $u_{51} = 2$. Due to $a_{12} = 4$, the residual availability is 2 capacity units.

It is well known that in order to obtain optimal solutions for RCPSP and GRCPSP instances, not all feasible schedules have to be examined (cf., e.g., Sprecher et al. (1995) and Demeulemeester and Herroelen (1997 a)). In order to separate those schedules which have to be considered in any case from the remaining ones, providing an appropriate classification may be useful. For the well-known job shop problem, such classifications have been thoroughly studied (cf., e.g., Conway et al. (1967), Baker (1974), French (1982), and Hutchison and Chang (1990)). Some initial work for RCPSP has been performed by Wiest (1964). However, no widely accepted classification has been available until the work of Sprecher et al. (1995) which is presented in the sequel. Recently, Nübel and Schwindt (1997) have extended this classification also considering related resource-constrained project scheduling problems as discussed in the Sections 3.3 and 3.4.

When searching for an optimal solution of an RCPSP or a GRCPSP instance, those schedules can be excluded for which assigning a starting time $SS'_j < SS_j$ to a single job j (*ceteris paribus*) results in a feasible schedule again. Such reductions of single starting times leaving the remaining ones unchanged are called left shifts:

Definition 5.5. A *local left shift* of a job j consists of a sequence of starting time reductions by one period with each intermediate schedule being feasible. If there exists a feasible starting time $SS'_j < SS_j$ for a job j which can not be obtained by a local left shift, assigning SS'_j to j represents a *global left shift*.

Based on these definitions, the following two classes of schedules may be distinguished:

Definition 5.6. Feasible schedules for which no further local left shifts are possible constitute the set of *semi-active* schedules. A subset of this set is built by the *active* schedules, i.e., such schedules for which no further global left shift can be performed.

Remark 5.2. At least one optimal schedule of an RCPSP instance is active. The same is true for GRCPSP due to the objective function also being regular (cf. Sprecher et al. (1995) for a corresponding proof). Therefore, the efficiency of solution procedures can considerably be increased by evaluating active schedules only.

Remark 5.3. An objective function of a project scheduling problem is considered to be *regular* when the following condition holds for two partial schedules PS and PS'. If in PS each job is assigned a starting time not larger than in PS', then also the objective function value of PS is not larger than the one of PS' (cf. Conway et al. (1967, p. 12) and Sprecher et al. (1995)). Hence, minimizing the project completion time represents a regular objective function. By the way of contrast, maximizing the net present value is a non-regular one (cf. Section 3.4.4). A general discussion on which objective functions considered in project scheduling are regular and which are not is contained in Nübel and Schwindt (1997) as well as Neumann et al. (1999).

Example. The schedule depicted in Figure 5.2 b) is derived from the one in Figure 5.2 a) by performing two local left shifts. First of all, job 6 is left shifted for one period. Subsequently, the same is done with job 4 for 2 periods. Since no further local left shifts are possible, the schedule obtained is semi-active. Globally left-shifting job 4 such that it starts at the time point $t = 5$, results in the active schedule given in Figure 5.2 c).

When discussing the properties of heuristics, also non-delay schedules are of interest.

Definition 5.7. In a *non-delay schedule*, there is no time point t (period $t+1$) at (in) which an eligible job j can be started in addition to the active ones ignoring the resource constraints for the periods $t+2, \dots, t+d_j$. The set of non-delay schedules for an instance is a subset of the active ones.

This property can easily be verified for a feasible schedule after performing the following transformation of the underlying problem instance. Each job j is replaced by a chain of d_j unit-time sub-jobs j_1, \dots, j_{d_j} which have resource demands u_{jr} and get starting times $SS_{j1}, \dots, SF_{jd_j} - 1$. If none of the unit-time jobs

can be globally left shifted in the corresponding schedule, it represents a non-delay one.

Remark 5.4. For some instances, the set of non-delay schedules may not contain an optimal solution (cf. Sprecher et al. (1995)). That is, by considering only such schedules, it can not be guaranteed that an optimal solution is found for an instance.

Example. The active schedule depicted in Figure 5.2 c) is not a non-delay one, because at $t = 0$ job 5 could be additionally started without violating the resource constraint in period 1. This can be verified by splitting job 5 into two unit-time jobs. Then, the first of the resulting jobs may be globally left shifted to $t = 0$. Obviously, the non-delay schedule of Figure 5.2 d) is not optimal.

5.2 Priority-Rule Based Methods

In this section, we describe different priority-rule based methods for solving RCPSP and GRCPSP. Though already a large amount of research has been performed on such methods for RCPSP, continuously examining concepts for their improvement is justified for the following reasons:

- Priority-rule based heuristics are in wide and general use due to yielding acceptable results with a reasonable computational effort. They can easily be understood and are transparent to the user thus being incorporated in many commercial project management software packages.
- Solutions obtained by priority-rule based procedures provide points of departure for improvement methods with their quality having considerable influence on the performance of such procedures (cf. Section 5.3). Though improvement methods usually yield better results than priority-rule based procedures, their implementation requires more skills as well as their execution more computational effort.
- The solution quality obtained by proprietary heuristics contained in software packages often is not satisfying (cf. Johnson (1992), Maroto and Thomas (1994), Farid and Manoharan (1996), Kolisch and Hempel (1996 b, c), Maroto et al. (1999), and Section 7.4.1.4). Since modern software packages provide the possibility of including own procedures comfortably either by specifying priority values (cf. Khattab and Soyland (1996)) or by incorporated programming languages (e.g., Visual Basic for Applications which is

a part of Microsoft Project), more efficient, easy to implement priority-rule based procedures are of great interest for project management software developers and users.

In general, a priority-rule based heuristic consists of two major components, the *scheduling scheme* and the *priority rule*. The scheduling scheme defines how a feasible schedule can be constructed by successively assigning scheduled starting times to jobs. In Section 5.2.1, two basic scheduling schemes are described for RCPSP and GRCPSP instances. In Section 5.2.2, we show for RCPSP, how these schemes can benefit from backward and bidirectional planning, an extension which is not possible for GRCPSP. While building a schedule, priority rules are applied to select the job to be scheduled next. Section 5.2.3 surveys the most important priority rules which have been proposed for RCPSP as well as introduces new ones. Furthermore, a brief discussion is included showing, how these rules can be adapted to GRCPSP. Note that up to now, no priority-rule based heuristics have been examined for GRCPSP at all!

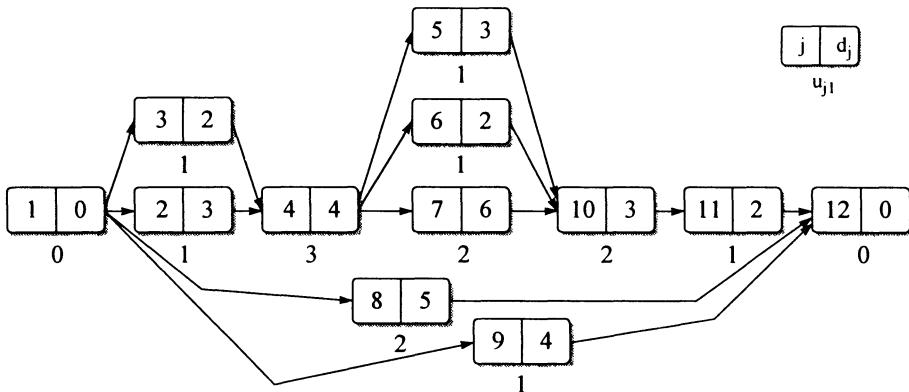


Figure 5.3. Example finish-to-start project network

In order to improve the solution quality yielded by a priority-rule based heuristic, often several combinations of a scheduling scheme and a priority rule are applied to a single problem instance. The resulting heuristics are commonly referred as a *multi-pass* priority-rule based heuristics in order to distinguish them from the *single-pass* ones restricting to the evaluation of a single combination. Different approaches of this type are discussed in Section 5.2.4.

To ease the discussion, we use the RCPSP example depicted in Figure 5.3. It shows a project with $n = 12$ jobs and one renewable resource type ($m = 1$)

which is available with $a_1 = 4$ units per period. A corresponding optimal solution is given in the capacity-oriented Gantt chart in Figure 5.4.

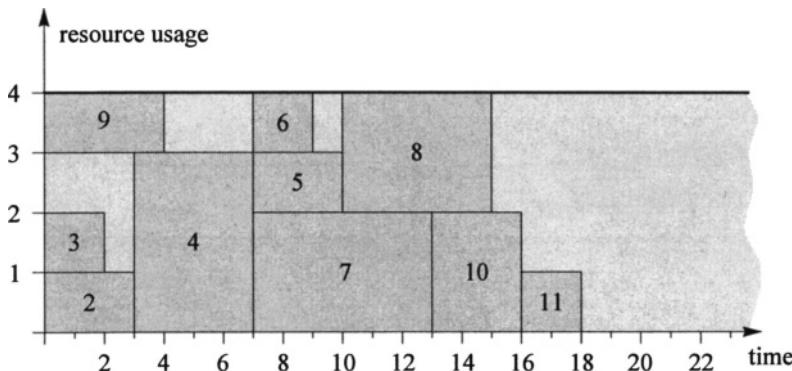


Figure 5.4. Optimal solution

5.2.1 Scheduling Schemes

In the following, the *serial scheduling scheme* (SSS) and the *parallel scheduling scheme* (PSS) are introduced which are most frequently used in priority-rule based heuristics for RCPSP (cf. Kolisch (1996 b)). Whereas the serial scheduling scheme can immediately be transferred to GRCPSP, the parallel scheduling scheme has to be slightly modified. For a more formal statement of the scheduling schemes see Kolisch (1995, section 5.2) and Kolisch (1996 b).

5.2.1.1 Serial Scheduling Scheme

The *serial scheduling scheme* (SSS) was proposed for RCPSP by Kelley (1963). It constructs a feasible schedule in exactly n iterations. In each iteration, the current partial schedule PS is extended by scheduling one job as follows. After determining the set $A(PS)$ of *available* jobs, one of these jobs is selected by a priority rule. This job j is scheduled to start at the earliest time point $t \geq ES_j(PS)$ for which its execution does not violate the resource constraints in the periods $t+1, \dots, t+d_j$. For the resulting partial schedule PS' with $J(PS') = J(PS) \cup \{j\}$, the residual availabilities are modified and the earliest starting and finishing times $ES_j(PS')$ and $EF_j(PS')$ of all unscheduled jobs are recalculated. In the next iteration, PS' is extended and so on until all jobs are scheduled. Applying SSS requires $O(m \cdot n^2)$ time (cf. Kolisch and Hartmann

(1999)) provided that this computational effort is not exceeded by the one for computing the priority values.

Example. Figure 5.5 shows the solution which is obtained for our example by applying SSS in forward direction stipulating that the shortest processing time rule (SPT) is used for selecting the job to be considered in each iteration. SPT always chooses the job having the shortest processing time (cf. Section 5.2.3). After having scheduled the dummy job 1 in the first iteration, the jobs 2, 3, 8, and 9 are available. Following SPT, the jobs 3 and 2 are assigned the starting time 0 in the next two iterations and their successor 4 enters the set of available jobs. Job 4 is selected and its starting time is set to $SS_4 = 3$ resulting in the jobs 5, 6, 7, 8, and 9 being available. Subsequently, we yield $SS_6 = 7$, $SS_5 = 7$, and $SS_9 = 0$. Now, only the jobs 7 and 8 are available among which job 8 is selected. However, it can not be started at its earliest starting time $ES_8(PS) = 0$ due to the resource constraint. The earliest time point t for which the residual availability in the following periods $t+1, \dots, t+d_j$ is not smaller than $u_{81} = 2$ is $t = 7$. For the remaining available job 7, observing the resource constraint leads to $SS_7 = 10$. Finally, the jobs 10 and 11 are scheduled consecutively yielding a schedule with a project completion time of 21 periods.

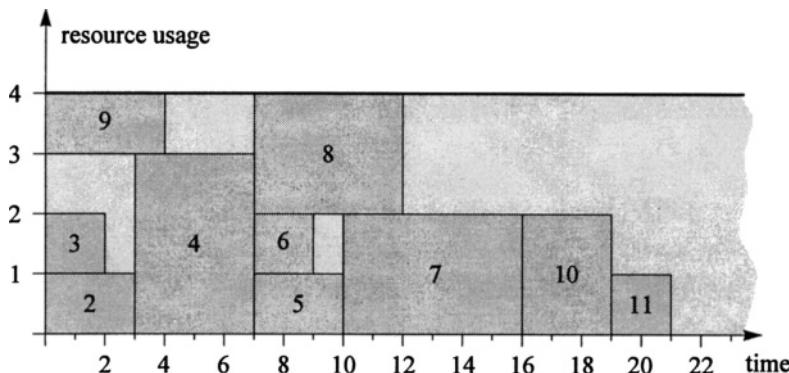


Figure 5.5. SSS (forward)

Remark 5.5. SSS constructs *active schedules* (cf. Kolisch (1995, section 5.2) and Kolisch (1996 b)). This can easily be verified. In each iteration, an available job is scheduled as early as possible without violating the precedence and the resource constraints. Hence, for the resulting schedule, no global left shift is possible.

Within single-pass heuristics, the effectiveness of SSS has been examined, e.g., by Müller-Merbach (1967 b), Fehler (1969), Gonguet (1969), Cooper (1976), Slowinski (1978), Boctor (1990), Kolisch (1995, section 5.2), Kolisch (1996 b), Drexel et al. (1997), and Klein (1998).

5.2.1.2 Parallel Scheduling Scheme

For RCPSP, the *parallel scheduling scheme (PSS)* generates a feasible schedule in at most n iterations (cf. Brooks and White (1965)). In each iteration, the current partial schedule PS with an associated *decision point* t (which terminates period t) is extended. This decision point is chosen such that all scheduled jobs $j \in J(PS)$ start before t , i.e., $SS_j(PS) < t$. At the beginning of each iteration, the set $E(PS,t)$ of (*forward*) *eligible jobs* containing all available jobs with $ES_j(PS) \leq t$ is determined. Subsequently, the following step is repeated until $E(PS,t)$ is empty. According to the priority rule used, a job $j \in E(PS,t)$ is selected and removed from $E(PS,t)$. If starting job j at time point t does not violate the resource constraints in the periods $t+1, \dots, t+d_j$, it is scheduled ($SS_j = t$) and the residual availabilities in the following periods are reduced appropriately. Otherwise, job j is discarded. At the end of each iteration, the next decision point t' for the resulting schedule PS' is computed. It corresponds to the minimum finishing time $SF_j(PS')$ of all jobs $j \in J(PS')$ being active in period $t+1$. The process stops when all jobs have been scheduled. According to SSS, the computational effort required by PSS is of the order $O(m \cdot n^2)$ (cf. Kolisch and Hartmann (1999)).

Example. In Figure 5.6, the solution yielded for the example of Figure 5.3 by PSS in combination with the priority rule SPT is given. After assigning $SS_1 = 0$ in the first iteration, we yield the decision point $t = 0$ and $E(PS,t) = \{2, 3, 8, 9\}$ for $J(PS) = \{1\}$. According to the SPT rule, the jobs 3, 2, and 9 are selected, removed from $E(PS,t)$ and scheduled at $t = 0$. The remaining eligible job 8 can not be started because of the residual availability of the single resource type not being sufficient in the periods 1 and 2. The second iteration is terminated with determining the next decision point $t' = \min\{SF_2, SF_3, SF_9\} = 2$. Still only job 8 is eligible and, hence, started at $SS_8 = 2$. After the completion of job 2 at $t = 3$, job 4 becomes eligible. However, due to the resource constraint it can not be scheduled in this and the subsequent iteration with the decision points $t = 3$ and $t = 4$, respectively. In the fifth iteration ($t = 7$), job 4 is assigned the starting time $SS_4 = 7$ resulting in

$t' = 11$ with $E(PS, t') = \{5, 6, 7\}$. These jobs can be executed concurrently and, hence are scheduled. Finally, the jobs 10 and 11 are processed consecutively, and we obtain a solution with a project completion time of 22 periods.

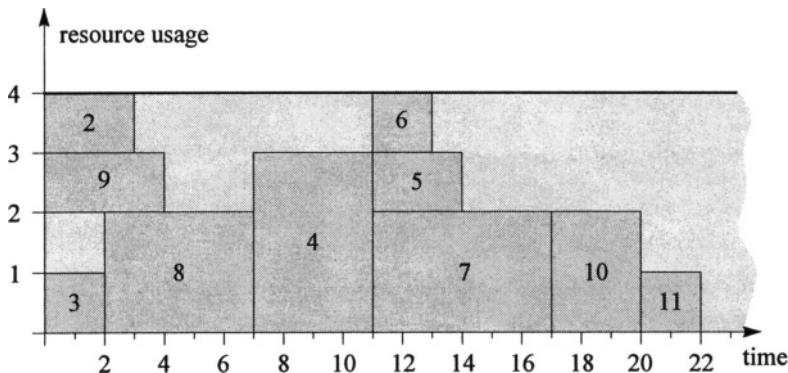


Figure 5.6. PSS (forward)

In order to apply PSS for GRCPSP, only the computation of the next decision point t' has to be modified. For RCPSP, no additional job can be started before the time point at which one of the jobs being active in period $t + 1$ is terminated because for the unscheduled jobs $j \in E(PS, t)$ the residual resource availabilities are not sufficient and no additional job can become eligible. However for GRCPSP, this assumption is no longer valid due to the prespecified release dates, the minimum time lags between jobs, and the fluctuations in the resource availabilities. Therefore, the next decision point t' is the smallest time point with $t' > t$ for which a job $j \in J - J(PS)$ can be started according to the modified forward pass, an active job is terminated or at which the resource availability of at least one resource type increases.

Remark 5.6. By applying PSS, *non-delay schedules* are constructed (cf. Kolisch (1995, section 5.2) and Kolisch (1996 b)): At each decision point t , the eligible jobs are considered in the order of their priority values. Each job $j \in E(PS, t)$ is scheduled if the residual resource availability is sufficient. Otherwise, it is discarded. Hence, in the resulting schedule there is no job which can additionally be started at a time point t .

Remark 5.7. Initially, the term parallel scheduling scheme has been introduced by Kelley (1963). However, the according algorithm differs from the one given above. In contrast to the above description, jobs which are started before

the current decision point t and are still active may be unscheduled again if they prevent jobs with a better priority value from being started at t . That is, this scheme is closely related to the branching scheme based on delaying alternatives employed by Demeulemeester and Herroelen (1992) (cf. Section 6.4.3, pp. 256). Following the argumentation of Sprecher and Drexl (1996), it can be shown that the application of the parallel scheduling scheme as described by Kelley (1963) does not result in semi-active schedules. As a consequence, the quality of solutions determined is rather poor which is confirmed by preliminary computational results of the author. Therefore, the above version of PSS is used. Furthermore, this is justified because nearly the complete literature on priority-rule based heuristics refers to this version (cf. Kolisch (1995, p. 68)).

Besides in Brooks and White (1965), PSS is employed by Bedworth (1973), Patterson (1973), Davis and Patterson (1975), Patterson (1976), Thesen (1976), Cooper (1977), Whitehouse and Brown (1979), Elsayed (1982), Elsayed and Nasr (1986), Ulusoy and Özdamar (1989), Boctor (1990), Khattab and Choobineh (1991), Özdamar and Ulusoy (1994), Ulusoy and Özdamar (1994), Kolisch (1996 a, 1996 b), Drexl et al. (1997), Thomas and Salhi (1997), and Klein (1998).

5.2.1.3 A Critique of the Scheduling Schemes

For our example, SSS combined with the SPT rule yields a better result than PSS with the same rule. However, the two scheduling schemes do not dominate each other in the sense that one always yields a better solution than the other one (cf., e.g., Cooper (1977) and Kolisch (1996 b)). This is underlined by the examples for backward planning given in the next section with PSS obtaining the better result. As a consequence, the question arises which of the two scheduling schemes should be used when implementing a heuristic for solving RCPSP and GRCPSP.

In this context, the statements of the Remarks 5.5 and 5.6 are important. The schedules constructed by (forward) SSS belong to the class of active schedules among which at least one optimal solution for each RCPSP and GRCPSP instance is contained. By the way of contrast, (forward) PSS only yields non-delay schedules. For some instances, this class may not contain an optimal solution at all, i.e., applying PSS can not lead to an optimal solution irrespective of which priority rule is used. However, anticipating the computational results in

Section 7.4.1.1, pp. 287, SSS performs slightly worse than PSS considering the average deviation from optimality but manages to find more optimal solutions, at least for RCPSP. In the following, we try to partially explain this contradictory behavior of the two scheduling schemes.

Consider the application of SSS and the SPT rule to the RCPSP example depicted in Figure 5.7 assuming a single resource type $r = 1$ with a per period availability of $a_1 = 4$. The finish-to-start project network consists of two disjoint chains of different lengths with no job of one chain being related to a job of the other one by precedence. The longer one of the two chains defining the critical path $CP = \langle 1, 5, 6, 7, 8, 9, 10 \rangle$ contains more jobs with durations shorter than the smallest one of a job on the non-critical chain. Only the jobs 6 and 8 may be processed in parallel to the non-critical ones 2, 3, and 4.

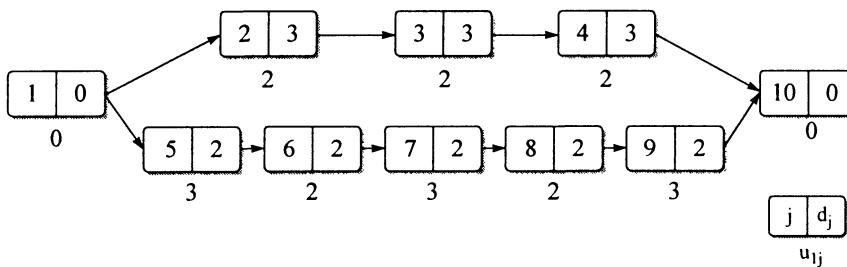


Figure 5.7. Example finish-to-start project network

Due to using the SPT rule, the critical jobs are all scheduled before any job of the other chain. Then, after having scheduled the critical jobs with durations of $d_j = 2$ consecutively, the intervals with residual availabilities allowing for processing any non-critical job $h \in \{2, 3, 4\}$ in parallel are always shorter than its duration $d_h = 3$. Thus, the non-critical jobs have to be performed after the critical ones and a schedule where all jobs are processed exclusively is obtained. The corresponding project completion time is $CT = 19$. For the optimal solution in which the job pairs (2,6) and (3,8) are processed partially in parallel, a project completion time of $CT = 15$ is yielded.

Though the example is extreme, it demonstrates the problem inherent to SSS, i.e., that partial schedules with a rather poor resource utilization may be obtained which again leads to solutions with a rather large completion time.

The problem described for SSS can not occur for PSS. At each decision point t , eligible jobs are added to the current partial schedule until this is no longer pos-

sible due to the residual resource availabilities being too small. However, the problem arises that the resources are (completely) absorbed at the current decision point t in a greedy manner. Scheduling jobs which can be delayed without leading to an increased project completion time may avoid the starting of jobs without this property which become eligible at one of the next decision points. For example, consider the schedule obtained by PSS in Figure 5.6, p. 172. As a consequence of assigning the starting time $SS_8 = 2$ to job 8, job 4 can no longer be scheduled at $t = 3$ and the optimal solution with a project duration of 18 is missed clearly.

5.2.2 Multiple Planning Directions

In the sequel, we show how the effectiveness of priority-rule based heuristics for RCPSP can be increased by employing different planning directions (cf. Klein (1998)). For GRCPSP, planning in multiple directions is not possible (cf. Remark 5.9).

5.2.2.1 *Backward Planning*

In the following, we explain how the scheduling schemes presented in the previous section can be used to construct a feasible schedule by planning in a backward direction. This idea has first been described for resource-constrained project scheduling by Li and Willis (1992) and has also successfully been applied by Özdamar and Ulusoy (1996 b, 1996 c).

The most simple approach to perform a backward planning consists in reversing the project network and applying the respective scheduling scheme to the resulting problem instance. Figuratively speaking, any project network can be "mirrored" by reversing all arcs such that a backward planning becomes a forward planning and vice versa. The results published so far reveal that changing planning directions has considerable effect on the performance of a heuristic.

Another concept explicitly adapts the scheduling schemes for backward planning which requires to extend some of the definitions made in Section 5.1. According to forward planning, partial schedules are successively extended by applying the scheduling schemes until a feasible solution has been obtained. However, the scheduling process starts with assigning a finishing time to the terminal end job n . This finishing time corresponds to a valid upper bound \bar{T} on the project completion time which can, e.g., be determined by the sum of all job

durations (cf. Section 3.2.1.1, pp. 77). A job is *backward available* if all its successors have been scheduled. By a *modified backward pass* latest finishing and starting times are computed setting $LF_j(PS) = SF_j$ and $LS_j(PS) = SS_j$ for all jobs $j \in J(PS)$.

Serial Scheduling Scheme. Within the *backward serial scheduling scheme*, a backward available job is selected in each iteration and scheduled as late as possible without violating the precedence and resource constraints. Usually, the schedule obtained will not start at the beginning of the planning horizon. Therefore, to "left shift" the schedule, the scheduled starting and finishing times $SS_j(PS)$ and $SF_j(PS)$ are reduced by subtracting the starting time $SS_1(PS)$ of the dummy start job. The project completion time yielded equals the resulting finishing time $SF_n(PS)$ of the dummy end job.

Example. The result of applying the backward SSS with the SPT rule to the example of Figure 5.3, p. 168, with $a_1 = 4$ is given in Figure 5.8. Scheduling is started from $\bar{T} = 34$ corresponding to the sum of all job durations. After completing the dummy end job, the jobs 11, 9, and 8 are backward available. Job 11 is assigned the finishing time $SF_{11} = 34$. Subsequently, job 10, which has become available in the meantime, is scheduled followed by its predecessors 6 and 5. Now, job 9 is selected and scheduled with $SF_9 = 34$. As a consequence, job 8 must not finish later than $t = 30$ due to the resource constraint. Finally, the jobs 7, 4, 3, and 2 are chosen and scheduled in this sequence yielding a schedule with $SS_1 = 13$. Reducing all scheduled starting times by an amount of 13 shifts the schedule to the beginning of the planning horizon resulting in a project completion time of 21.

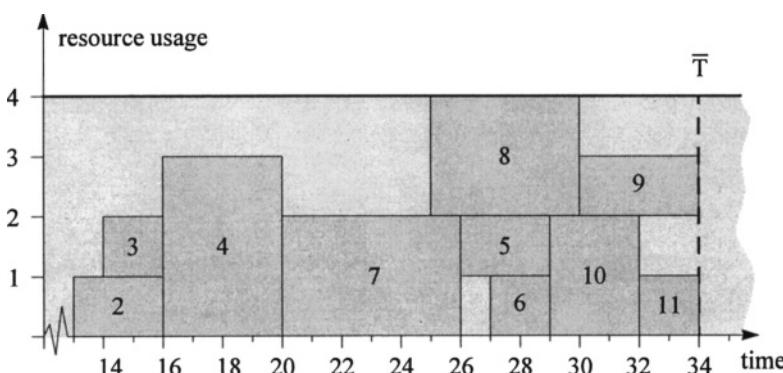


Figure 5.8. SSS (backward)

Parallel Scheduling Scheme. In each iteration, the *backward parallel scheduling scheme* considers a partial schedule PS with a decision point t such that all jobs $j \in J(PS)$ finish after t , i.e., $SF_j(PS) > t$ for all $j \in J(PS)$. The set of *backward eligible* jobs contains all backward available jobs with $LF_j(PS) \geq t$. The jobs out of this set are examined according to the priority rule and, if this does not lead to a violation of the resource constraints, scheduled at $SF_j = t$. At the end of each iteration, the next decision point t' is set to the maximum starting time $SS_j(PS)$ of all jobs being active in period t . After computing a feasible schedule, it is transformed to start at the beginning of the planning horizon as described for backward SSS.

Example. Figure 5.9 presents the solution obtained by backward PSS and the SPT rule. Having scheduled the dummy end job 12, the jobs 11, 9, and 8 which are backward eligible at $t = 34$ can be executed in parallel. Due to the resource constraint, job 10 is scheduled such that it finishes at $t = 30$. At the decision point $t = 27$, the jobs 5, 6, and 7 become backward eligible and, hence, are performed in parallel finishing at $t = 27$. Finally, the jobs 4, 3, and 2 are scheduled. By reducing all the scheduled starting and finishing times by 14, a schedule with a completion time of 20 is obtained, i.e., the schedule yielded by the (forward) PSS is improved.

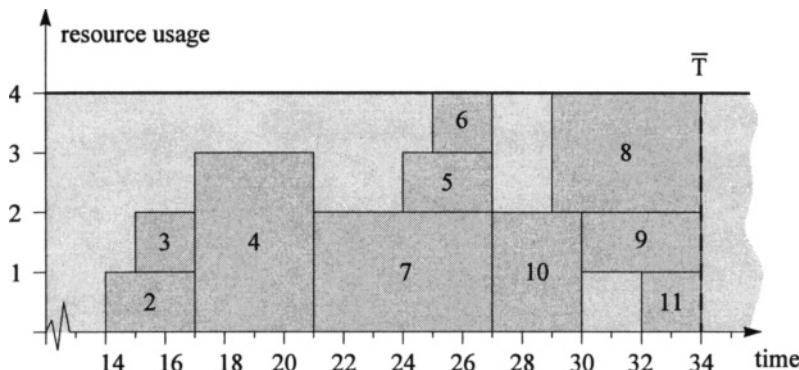


Figure 5.9. PSS (backward)

Remark 5.8. Usually, project managers want jobs to start as soon as possible in order to have as much slack available as possible in case of the execution of one or more jobs being delayed. To obtain schedules with jobs having this property the transformation described above is not sufficient. In the example for backward PSS (Figure 5.9), the jobs 5 and 6 are assigned the starting times

$SS_5(PS) = 24 - 14 = 10$ and $SS_6(PS) = 25 - 14 = 11$, respectively. However, they could already be started after having finished job 4 at the time point $SF_4(PS) = 21 - 14 = 7$. In Section 5.2.2.2, we propose a method for reducing the scheduled starting and finishing times such that all jobs are processed as soon as possible while preserving the feasibility of the schedule. Note that by this transformation the project completion time is not affected.

Remark 5.9. For GRCPSP, backward planning can not be applied. Due to the fluctuations in the resource availabilities, different schedules are obtained when starting with scheduling the dummy end job at different time points \bar{T} . Examining all possible project completion times as points of departures is prohibitive.

5.2.2.2 Bidirectional Planning

Besides performing forward and backward planning, the scheduling schemes can be extended to assign starting times *bidirectionally*, i.e., to construct schedules in forward and backward direction simultaneously. This is achieved by computing forward and backward priority values for all jobs, respectively (cf. Section 5.2.3). The idea of bidirectional planning has already successfully been applied to the job-shop problem (cf. Dell'Amico and Trubian (1993)) and the simple assembly line balancing problem (cf. Klein and Scholl (1996), Scholl and Voß (1996), Scholl and Klein (1997), and Scholl (1999, chapters 4 and 5)).

Serial Scheduling Scheme. The *bidirectional serial scheduling scheme* considers a forward partial schedule FS and a backward one BS simultaneously. In each iteration, a job j is selected among those available in forward, backward, or both directions. No decision upon the planning direction has to be made if job j is only available in one direction. Otherwise, it is scheduled in the direction for which the priority value leading to its selection has been computed. Since for some priority rules the planning direction may still not be determined, e.g., in case of SPT, the following tie break rule is applied. If $SS_j(FS) \leq \bar{T} - SF_j(BS)$, job j is added to FS otherwise to BS. This rule follows the idea that it is advantageous to schedule a job as soon as possible in forward direction and as late as possible in backward direction, respectively. If ties are still not broken, job j is scheduled in forward direction.

Having obtained a forward and a backward schedule, there still remains the problem of *interlinking* these two schedules to a single one. For this purpose, the jobs $j \in J(BS)$ are considered in non-decreasing order of their scheduled

starting times $SS_j(BS)$. Following this order, the jobs are successively *left shifted* into FS, i.e., assigned the smallest possible starting time which can be realized observing the precedence and the resource constraints. In case of several jobs having identical starting times, the jobs are considered in the reverse order in which they have been scheduled, i.e., the job scheduled the last is left shifted first.

Example. Figure 5.10 shows the forward and the backward schedule yielded by the bidirectional SSS in combination with the SPT rule when applied to our example defined by Figure 5.3, p. 168 with $a_1 = 4$. After scheduling the dummy start and end jobs, the jobs 2, 3 are forward available and the job 11 is backward available, while the jobs 8 and 9 are available in both directions. Following SPT, the jobs 3 and 11 are added to FS and BS, respectively. Subsequently, job 2 extends FS and job 10 which has become backward available in the meantime is added to BS. Due to having BS extended by job 10, also the jobs 6 and 5 are added to BS, because they have smaller processing times than the other available jobs 4, 7, 8 and 9. In the next iteration, job 9 is selected which is available in both directions. Applying the tie break rule yields $SS_9 = 0$. After starting job 4 at $SS_4 = 3$, the only unscheduled jobs are 7 and 8 among which job 8 is chosen. Ties are broken in favor of backward planning. The same is true for job 7.

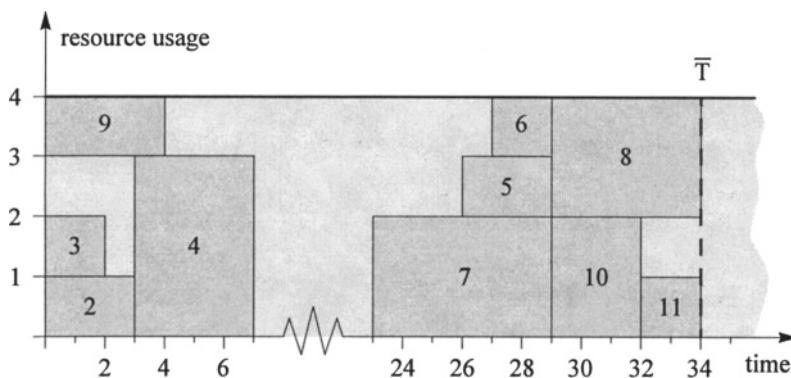


Figure 5.10. SSS and PSS (bidirectional)

By interlinking BS and FS the optimal solution given in Figure 5.4, p. 169, is obtained as follows. First of all, the starting times of the jobs 7, 5, and 6 are reduced to 7. Subsequently, job 8 is left shifted as far as possible observing the resource constraints ($SS_8(FS) = 10$). Job 10 can not start before $t = 13$ due to

the precedence constraints. Finally, job 11 is started at $SS_{11}(FS) = 16$. That is, by bidirectional planning an optimal solution for the example problem is determined.

Parallel Scheduling Scheme. The *bidirectional parallel scheduling scheme* simultaneously extends a forward schedule FS and a backward one BS with decision points ft and bt, respectively. At the beginning of each iteration, the sets of forward eligible jobs $FE(FS, ft)$ and backward eligible jobs $BE(BS, bt)$ are calculated. In each of the subsequent steps, a job $j \in FE(FS, ft) \cup BE(BS, bt)$ is selected by a priority rule and removed from the set(s). The planning direction is determined according to bidirectional SSS resulting in job j being scheduled either to start at ft or to finish at bt provided that the residual availabilities of the resource types are sufficient. Otherwise, job j is discarded for the current iteration. The iteration stops as soon as one of the two sets is empty. For the planning direction whose set of eligible jobs has been completely considered, the next decision point is calculated. Having scheduled all jobs, FS and BS are interlinked as described above.

Example. By applying the bidirectional PSS in combination with the SPT rule to the example of Figure 5.3, p. 168, with $a_1 = 4$, we obtain the same forward and backward schedule as for SSS (cf. Figure 5.10). However, the jobs are considered in another sequence. In the first iterations, the dummy jobs are scheduled and we obtain $FE(FS, ft) = \{2, 3, 8, 9\}$ and $BE(BS, bt) = \{8, 9, 11\}$ with $ft = 0$ and $bt = 34$. In the following steps, we assign the starting and finishing times $SS_2(FS) = 0$, $SF_{11}(BS) = 34$, $SS_3(FS) = 0$ and $SS_9(FS) = 0$. The jobs 2, 11, and 3 are only eligible in a single direction, respectively. Job 9 is added to FS due to the tie breaking rule applied. Subsequently, the remaining eligible job 8, which is forward and backward eligible, can only be scheduled in backward direction and, hence, $SF_8(BS) = 34$. Now, the iteration stops yielding the new decision points $ft = 2$ and $bt = 32$, respectively. Due to $FE(FS, ft) = \{\}$, ft is increased to 3 and the jobs 4 and 10 are scheduled in the following two iterations. In the last iteration, the jobs 5, 6, and 7 are eligible in both planning directions with ties being broken in favor of backward planning.

Remark 5.10. Note that by bidirectional planning a major disadvantage of PSS can be overcome. The optimal solution given in Figure 5.4 does not belong to the class of non-delay schedules and, hence, can not be obtained by the traditional PSS (cf. Kolisch (1996 b)). However, it is yielded by applying PSS bidirectionally.

5.2.3 Priority Rules

As described previously, priority rules are applied within a scheduling scheme for selecting one among several jobs which is scheduled next. For this purpose, a priority value p_j is calculated for each job $j \in J$. Depending on the rule, one with the smallest or largest value is chosen. In case of several jobs having the same value, a further rule may be applied as tie breaker. For RCPSP, for which backward planning is possible in contrast to GRCPSP, we omit giving the corresponding versions of the priority rules, because they can easily be derived following the principle of reversing the project network.

Types of Priority Rules. Many different priority rules for RCPSP have been presented in the literature so far. When performing the computational study presented in Section 7.4.1, we found and evaluated a total of 73 priority rules. Only the most promising and most frequently used rules are described in the following. For further priority rules, we refer to the literature given for SSS and PSS in Section 5.2.1. Depending on the information used by a rule, different types of rules may be classified as follows (cf., e.g., Alvarez-Valdés and Tamarit (1989 b) and Kolisch (1995)):

- *Network based rules* exploit information contained within the project network thereby discarding resource data. Examples are the job durations, the number of successors and so on.
- *Critical path based rules* are based on the results of a forward and a backward pass. Most commonly, they refer to earliest and latest starting and finishing times.
- *Resource based rules* refer to the resource requirements of jobs. For example, they may compute the average relative amount a job requires of all resource types.
- *Composite rules* try to overcome the disadvantage of relying on a single type of information. For this purpose, they usually combine network, time and resource based rules by computing a weighted sum of the priority values obtained by the respective rules.

A further distinctive feature of priority rules concerns the frequency with which they have to be computed. *Static rules* calculate priority values only once before starting the scheduling process. By the way of contrast, *dynamic rules* update the priority values each time a job has been scheduled. Some rules can be

Table 5.2. Priority rules

rule	extr.	priority value p_j
SPT	min	d_j
MIS	max	$ F_j $
MTS	max	$ F_j^* $
RPW	max	$d_j + \sum_{i \in F_j} d_i$
RPW*	max	$d_j + \sum_{i \in F_j^*} d_i$
EST	min	ES_j
EFT	min	EF_j
ESTD	min	$ES_j(PS)$
EFTD	min	$EF_j(PS)$
LST	min	LS_j
LFT	min	LF_j
MSL	min	$LS_j - ES_j$
MSLD	min	$LS_j(PS) - ES_j(PS)$
GRD	max	$d_j \cdot \sum_{r=1}^m u_{jr}$
WRUP	max	$\omega \cdot F_j + (1-\omega) \cdot \sum_{r=1}^m (u_{jr}/a_r)$
ACROS	max	$\max \{ \sum_{i \in \Pi_{jh}} \sum_{r=1}^m (u_{ir}/a_r) \mid h = 1, \dots, \pi_j \} =: acr_j$
ACTRES	max	$\max \{ \sum_{i \in \Pi_{jh}} \sum_{r=1}^m (u_{ir} \cdot d_i/a_r) \mid h = 1, \dots, \pi_j \} =: act_j$
TIMROS	max	$\omega \cdot (LS_n - LS_j)/(LS_n - LS_1) + (1-\omega) \cdot acr_j/acr_1$
TIMRES	max	$\omega \cdot (LS_n - LS_j)/(LS_n - LS_1) + (1-\omega) \cdot act_j/act_1$
IRSM	min	$\max \{ 0, \max \{ ES_i^{ra}(PS, j) - LS_i \mid i \in E(PS, t) - \{j\} \} \}$
WCS	min	$LS_j(LBC1) - \max \{ ES_i^{ra}(PS, i) \mid i \in E(PS, t) - \{j\} \}$
ACS	min	$LS_j(LBC1) - \sum_{i \in E(PS, t) - \{j\}} ES_i^{ra}(PS, i) / (E(PS, t) - 1)$
WCLS	min	$\min \{ LS_i^{ra}(PS, i) \mid i \in A(PS) \}$
RAND	max	$rand()$

applied either statically or dynamically whereas others can be used in both fashions. For example, critical path rules may depend on values obtained by an initial forward and backward pass or on values computed for the current partial schedule. Among composite rules, the *regret based rules* constitute a separate class. Such dynamic rules determine a priority value (regret) assuming that a job j is not scheduled next, i.e., that another job is selected instead of j .

Table 5.2 presents the priority rules considered within our computational experiments. The selection has been made on the basis of preliminary computational results identifying the most successful rules of each type. In the first column, the abbreviation for each rule is given. The second column (extremum) denotes whether the rule selects a job with the smallest (min) or the largest (max) priority value p_j . The last column shows how the priority value p_j of a job j is computed. The notation which has not been introduced so far is defined when explaining the corresponding rules.

Network Based Rules. The Table 5.2 is subdivided into five parts. The first part contains some network based priority rules which are all static. The *shortest processing time rule* (SPT) which is well-known from a number of machine scheduling problems considers the job durations. The *most immediate successors rule* (MIS) and the *most total successors rule* (MTS) are based on counting the followers of each job. The *greatest rank positional weight rules* (RPW and RPW*) combine both characteristics. For further network based rules, see, e.g., Alvarez-Valdés and Tamarit (1989 a, 1989 b).

Critical Path Based Rules. In the second part of Table 5.2, the most important critical path based rules are given. The *earliest starting and finishing time rules* (EST, EFT, ESTD, and EFTD) are considered in a static and a dynamic version, respectively. The same is true for the *minimum slack time rule* (MSL). Within the parallel scheduling scheme, applying the *dynamic minimum slack time rule* (MSLD) yields the same results as the latest starting time rule due to the earliest starting times of all eligible jobs being equal to the current decision point t . In contrast to considering only a single planning direction, the *latest starting and finishing time rules* (LST, LFT) can also be computed dynamically when scheduling bidirectionally. For this purpose, the latest starting and finishing times for FS are calculated by considering BS and vice versa. The computational results in Section 7.4.1 refer to the dynamic versions of these rules.

Resource Based Rules. The third part contains the *greatest resource demand rule* (GRD) as well as a simple composite rule (*weighted resource utilization and precedence*, WRUP) which has been proposed by Ulusoy and Özdamar (1989). According to their proposal, we choose $\omega = 0.7$. Extensions of this rule have been presented by Thomas and Salhi (1997).

Composite Rules. The four static composite rules in the next part have initially been designed for considering only a single resource type and therefore have been extended. They all rely on determining a longest path from a job j to the dummy end job n in the project network the length of which gives the priority value p_j . However, the node weights w_j used for calculating these longest paths do not correspond to the durations of the jobs as this is, e.g., the case within the critical path based rules.

ACROS which has initially been introduced by Bedworth (1973) can be described as follows. For each job j , all paths $h = 1, \dots, \pi_j$ connecting job j and the dummy end job n are determined. The set of jobs which define a path h are denoted by Π_{jh} . That is, the length of path h can be computed by simply summing up the weights w_i of all jobs $i \in \Pi_{jh}$. Considering only a single resource type, Bedworth (1973) proposed to take the per period resource usage of a job as node weight. To extend this rule, we can calculate the average relative resource usage $w_i = \frac{1}{m} \cdot \sum_{r=1}^m u_{ir}/a_r$ of a job i . However, since the number m of resource types is a constant, we use $w_i = \sum_{r=1}^m u_{ir}/a_r$. Following the logic of the greatest resource demand (GRD) rule, Elsayed and Nasr (1986) have developed ACTRES which modifies ACROS by choosing weights according to the total resource usage necessary for processing a job j . As with ACROS, a version considering more than one resource type is obtained by calculating the relative total resource usage $w_i = d_i \cdot \sum_{r=1}^m u_{ir}/a_r$.

The rules TIMROS and TIMRES combine the rules ACROS and ACTRES with the rule ACTIM also introduced by Bedworth (1973). ACTIM determines the longest path from a job j to the dummy end job using job durations as node weights, i.e., $w_i = d_i$. However, it can easily be transformed into the latest starting time rule due to the length of the longest path connecting a job j and the dummy end job n being equal to $p_j = LS_n - LS_j$. Therefore, this rule is not considered explicitly. In order to combine the priority values of ACTIM with those of ACROS and ACTRES, respectively, the priority values calculated by

each rule are normalized by dividing them through the largest value which occurs for any job. Due to the structure of these rules, this value always corresponds to the one obtained for the dummy start job. Furthermore, the influence of the priority rules to be combined is controlled by building a weighted sum. In our computational experiments, we use $\omega = 0.5$.

Regret Based Rules. The next group consists of dynamic priority rules. The first three rules are designed to work with PSS and have been presented by Kolisch (1996 a). All rules have in common that the effect of selecting one job before the others is examined by considering pairs of eligible jobs. Consider two eligible jobs i and j . If job i is scheduled at the current decision point t , the residual resource availabilities in the periods $t+1, \dots, t+d_i$ are reduced by the demands of job i . Hence, it may no longer be possible to start an eligible job j at the current decision point t without violating the resource constraints. For example, consider the partial schedule PS in Figure 5.6, p. 172, with $J(PS) = \{2, 3\}$. For PSS, the jobs 8 and 9 are contained in the set of eligible jobs for the decision point $t = 0$. If job 8 is scheduled, job 9 can start at $t' = 2$ the earliest due to the resource constraints. Vice versa, the same is true for job 8, if job 9 is added to PS first. In general, we denote the earliest starting time which can be realized for an eligible job j after scheduling another eligible job i by $ES_j^{ra}(PS, i)$. The superscript "ra" indicates that besides the precedence constraints also the residual resource availabilities are considered for determining $ES_j^{ra}(PS, i)$.

A way for efficiently computing $ES_j^{ra}(PS, i)$ in case of RCPSP is described in Kolisch (1996 a). The value of $ES_j^{ra}(PS, i)$ corresponds to the smallest time point $q \geq t$ for which the aggregated resource demand of job i and job j does not exceed the residual availability of any resource type in the period $q+1$. Note that the periods $q+2, \dots$ do not have to be examined due to the constant availabilities of resources.

The *improved resource scheduling method* (IRSM) improves the classical resource scheduling method as proposed by Brand et al. (1964). For each partial schedule PS, the earliest schedule dependent starting time of the dummy end job $ES_n(PS)$ defines a simple lower bound LBC1 on the minimum project completion time (cf. Section 4.1.1.1, pp. 114). The basic idea of IRSM is to schedule an eligible job next for which the lower bound value $ES_n(PS')$ of the resulting partial schedule PS' does not increase at all or is as small as possible. This job is determined as follows. First of all, performing a backward pass

starting with $LF_n = LS_n = LBC1$ (current lower bound) yields latest finishing times $LF_j(LBC1)$ and starting times $LS_j(LBC1)$ until which jobs have to be terminated (started) the latest in order not to increase $LBC1$. Now, each eligible job $i \in E(PS, t)$ is scheduled by way of trial at the current decision point t and the residual resource availabilities of the periods $t + 1, \dots, t + d_i$ are reduced accordingly. Subsequently, the resulting values $ES_j^{ra}(PS, i)$ are calculated for all other eligible jobs $j \in E(PS, t) - \{i\}$. If for any job j the following condition $ES_j^{ra}(PS, i) > LS_j(LBC1)$ is fulfilled, scheduling job i before job j leads to an increase of $LBC1$ by $ES_j^{ra}(PS, i) - LS_j(LBC1)$ periods. Hence, the increase of $LBC1$ which results from scheduling job i next can be computed by the formula $p_i = \max \{0, \max \{ES_j^{ra}(PS, i) - LS_j(LBC1) \mid j \in E(PS, t) - \{i\}\}\}$. After determining this value, job i is released again and the priority values of the remaining eligible jobs are determined in the same fashion. Finally, an eligible job with the smallest priority value p_i is chosen and really scheduled.

The *worst case slack time rule* (WCS) extends the minimum slack time (MST) rule by computing a regret for not scheduling an eligible job $j \in E(PS, t)$ next. This regret corresponds to the smallest slack time of job j in a partial schedule PS' resulting from selecting another job instead of j (cf. Kolisch (1996 a)). To compute this value for job j , each other eligible job $i \in E(PS, t) - \{j\}$ is scheduled at the current decision point t by way of trial and the resulting value $ES_j^{ra}(PS, i)$ is computed. The maximum of all values $ES_j^{ra}(PS, i)$ (over i) represents the worst case earliest starting time of job j in case another job is scheduled next at decision point t . Accordingly, the worst case (smallest) slack time can be determined by calculating the difference between LS_j and this maximum value defining the priority value p_j . Among all eligible jobs, one with the smallest value p_j is chosen. Alternatively, instead of considering the worst case slack time of a job j , its *average case slack time* can be used as priority value (ACS).

The *worst case latest starting time rule* (WCLS) is designed to overcome the problem inherent to SSS sketched in Section 5.2.1.3. It has been inspired by the success of the LST rule which comes out first for SSS when the latter is combined with the traditional rules in the computational experiments described in Section 7.4.1.1, pp. 287. As described above, latest finishing times $LF_j(LBC1)$ and starting times $LS_j(LBC1)$ can be computed depending on the current partial schedule PS and the current lower bound $LBC1 = ES_n(PS)$. However, due to scheduling an available job i it may no longer be possible to

realize the latest starting time of another available job j observing the resource constraints. Hence, the *worst case latest starting time* of an available job j is computed as follows. Each other available job i is scheduled temporarily according to SSS and the resulting latest starting time $LS_j^{ra}(PS, i) \leq LS_j(LBC1)$ of job j with respect to the project duration $LBC1$ is calculated taking into account the residual resource availabilities. If, as a result of scheduling job i , no feasible starting time not larger than $LS_j(LBC1)$ exists for job j at all, we set $LS_j^{ra}(PS, i) = 0$. The smallest value obtained for all other available jobs i defines the priority value (worst case latest starting time) p_j . Following the logic of the LST rule, the job with the smallest priority value is selected among all available ones.

Finally, we include a *random rule* (RAND) which determines priority values randomly thus providing a benchmark for the priority rules based on logical reflections.

Priority Rules for GRCPSP. All priority rules presented for RCPSP can be transferred to GRCPSP stipulating that the earliest and latest starting and finishing times are computed by the appropriate forward and backward pass, respectively (cf. Section 3.2.2.1, pp. 91). This is in particular true for the network based, the critical path based, and the regret based rules as well as for the GRD rule which all can be applied as given in Table 5.2. Of course, when calculating the values $ES_j^{ra}(PS, i)$ and $LS_j^{ra}(PS, i)$ within the regret based methods, the fluctuations of the resource availabilities have to be taken into account. The computation of the WRUP rule as well as of the composite rules are based on the assumption that the per period availability is constant for each resource type. According to the computation of lower bound arguments, this requirement can partially be fulfilled by assuming a constant resource availability of $a_r^{\max} = \max\{a_{rt} \mid t = 1, \dots, \bar{T}\}$ for each resource type r (cf. Section 4.3.1, pp. 150). However, the effectiveness of these rules is reduced considerably as preliminary computational experiments have revealed.

5.2.4 Multi-Pass Priority-Rule Based Heuristics

The results presented in the Sections 7.4.1.1 and 7.4.1.2 reveal that the structure of problem instances has a considerable influence on which combination of scheduling scheme, planning direction and priority rule performs best. However, since computation times required by priority-rule based approaches are usu-

ally negligibly small, *multi-pass procedures* have been proposed for RCPSP in the literature. Such procedures are based on the idea that data dependency can partially be excluded by applying different combinations to the same problem instance and realizing the best solution obtained. In the literature, several types of multi-pass procedures can be distinguished (cf., e.g., Kolisch and Hartmann (1999)).

Multi-Priority Rule Methods. The basic idea of these methods is to combine one or both scheduling schemes with different priority rules in order to obtain different solutions. Two basic approaches can be distinguished. The first approach consists in applying mutually exclusive priority rules, e.g., the RPW* rule and the LST rule. This approach in combination with PSS is pursued by Patterson (1973), Boctor (1990), and Khattab and Choobineh (1991). In Klein (1998) both scheduling schemes are employed with different priority rules (cf. Section 7.4.1.3, pp. 293). The second approach relies on using a composite rule which computes a priority value as a weighted sum of values obtained by some network based, critical path based or resource based rules. By varying the weights at each application of the scheduling scheme, different solutions can be obtained. Corresponding methods for PSS are, e.g., proposed by Whitehouse and Brown (1979), Elsayed (1982), Elsayed and Nasr (1986), and Ulusoy and Özdamar (1989).

Multi-Planning Directions Methods. Such methods apply a combination of a scheduling scheme and a priority rule in different planning directions. In the methods proposed by Li and Willis (1992) for SSS and Özdamar and Ulusoy (1996 b, 1996 c) for PSS, forward and backward scheduling is performed by turns. The priority rule employed uses information on the schedule computed in the prior pass. In Klein (1998) different planning directions including bidirectional planning are combined with a selection of priority rules thereby defining a multi-priority rule, multi-planning direction method (cf. Section 7.4.1.3, pp. 293).

Sampling Methods. In general, these methods use a scheduling scheme together with a single priority rule. Depending on its priority value, a selection probability π_j is computed for each job j . In order not to obtain identical solutions in different passes, the next job to be scheduled is always chosen randomly with the probability of selecting a certain job j being equal to π_j . The number of passes performed is referred to as *sample size*. Two major approaches may be distinguished concerning the way of determining the selection probability π_j

of a job j . In the following these approaches are briefly described assuming that a priority rule is used which selects the job with the highest value p_j . For a more comprehensive description of sampling methods, we refer to Kolisch (1995, section 5.3.2).

- *Biased Random Sampling.* The basic idea of this approach is to bias the selection probability π_j of a job j according to its priority value, i.e., to calculate π_j directly depending on the priority value p_j . For this purpose, in case of using PSS, the following transformation $\pi_j = p_j / \sum_{h \in E(PS, t)} p_h$ is performed for each job $j \in E(PS, t)$ with the sum of the computed selection probabilities being equal to one. When SSS is applied, the transformation refers all available jobs $j \in A(PS)$. According procedures have, e.g., been considered by Wiest (1967) and Alvarez-Valdés and Tamarit (1989 a) for PSS and by Cooper (1976) for SSS. An extended version of biased random sampling is presented by Schirmer and Riesenber (1997 b).
- *Regret Based Biased Random Sampling.* Basically, this method corresponds to biased random sampling except for the difference that the selection probabilities are determined by using *regret values* which consider the priority values only implicitly. In case of using PSS, for each job $j \in E(PS, t)$, the *regret value* ρ_j is obtained by $\rho_j = p_j - \min\{p_h \mid h \in E(PS, t)\}$ whereas in case of applying SSS the computation is performed considering all available jobs $A(PS)$. Subsequently computing selection probabilities π_j based on ρ_j according to biased random sampling results in all eligible (available) jobs with the minimum priority value getting a selection probability of 0. Hence, the regret values are previously modified by calculating $\rho'_j = (\rho_j + \varepsilon)^\alpha$ with a small $\varepsilon > 0$ (cf. Drexel (1991)). Using the parameter α , the degree of the bias can be controlled. For $\alpha = 0$, all eligible (available) jobs have regret values of $\rho'_j = 1$, such that identical selection probabilities are obtained. By the way of contrast, for $\alpha \rightarrow \infty$, only the jobs with maximum regret values are assigned selection probabilities larger than 0. For RCPSP, regret based biased random sampling methods based on PSS as well as SSS have been examined by Kolisch (1995, sections 5.3 and 5.4), Kolisch (1996 b), and Schirmer and Riesenber (1997 b).

Adaptive Sampling Methods. In general, such approaches are based on sampling methods. Furthermore, they rely on the observation that various combinations of scheduling schemes and priority rules perform differently well depending on the problem instance present. The basic idea now consists of applying

the sampling method which will probably yield the best result for a problem instance on hand. For this purpose, instances are classified using the complexity measures described in Section 7.2.1, pp. 263. Subsequently, for each class of problem instances the appropriate sampling method is identified by computational experiments. Approaches following this idea have been presented by Kolisch and Drexel (1996) and Schirmer and Riesenber (1997 a). However, these approaches bear the danger of being "fine-tuned" only with respect to the benchmark data sets available. To overcome this possible disadvantage, Schirmer (1998) proposes an approach which selects the sampling method by employing case based reasoning (cf., e.g., Kimms (1998)).

5.3 Improvement Methods

As indicated at the beginning of this chapter, *improvement methods* are based on successively transforming a current solution into another one. By performing such transformations iteratively, the space of feasible solutions is systematically evaluated solution by solution in order to improve on an initial one which has, e.g., been determined by applying a priority-rule based heuristic.

Considering a minimization problem like RCPSP, *steepest descent* procedures only accept transformations which lead to an improvement of the *incumbent*, i.e., the best solution currently known. This restriction often results in determining a local optimum with a non-satisfying solution quality. In order to overcome this problem, also solutions representing an intermediate deterioration of the objective function value have to be examined. However, this bears the danger that a locally optimal solution is re-visited during the search process and certain regions of the solution space are examined repeatedly whereas others are not inspected at all. Due to this reason, modern heuristic search methods such as *simulated annealing* and *tabu search* have been introduced (cf. Sections 5.3.3 and 5.3.1, respectively). These improvement methods represent *meta-heuristics* which can be described as master strategies for controlling and guiding other heuristics in order to find solutions beyond those usually determined by identifying a local optimum. Another major group of meta-heuristics is constituted by the *evolutionary algorithms* which differ from improvement methods by modifying and combining a collection of solutions instead of transforming only a single one in each step of the search process. Within this group, *genetic algorithms* are most popular (cf. Section 5.3.3). Other recent meta-heu-

ristics are, e.g., neural networks (cf., e.g., Hopfield and Tank (1985) and Mason and Wang (1990)) and ant systems (cf., e.g., Colomi et al. (1991), Dorigo et al. (1996), and Sondergeld and Voß (1997)). For a classification of different meta-heuristics see, e.g., Pirlot (1996) and Glover and Laguna (1997, section 1.9).

This section is outlined as follows. To ease the description of a new tabu search procedure for GRCPSP (and RCPSP) in Section 5.3.2, a brief introduction into this meta-strategy is previously provided in Section 5.3.1. Finally, solution procedures proposed for solving RCPSP based on meta-heuristics are discussed in Section 5.3.3. This also includes approaches based on simulated annealing and genetic algorithms.

5.3.1 The Meta-Heuristic Tabu Search

In the sequel, a brief introduction into the meta-heuristic tabu search is presented which has initially been proposed by Glover (1989, 1990). Thereby, only those components are described the application of which seems to be most promising in order to develop efficient heuristic solution procedures for resource-constrained project scheduling. For more comprehensive descriptions we refer to De Werra and Hertz (1989), Voß (1993), Domschke et al. (1996 a, 1996 b), Aarts and Lenstra (1997) as well as Glover and Laguna (1997).

5.3.1.1 Moves, Neighborhood, and Descent Procedures

The basic purpose of improvement procedures like tabu search consists in efficiently examining the *solution space* of a problem defined by all its feasible solutions in order to determine a solution with a satisfying quality, i.e., a near-optimal objective function value. As described above, this is achieved by transforming a current solution \mathbf{x} into another one \mathbf{x}' and repeating this process for a number of iterations. The corresponding operations which are necessary to perform such a transformation are commonly termed as *moves*. Within resource-constrained project scheduling, the current solution may, e.g., be represented in form of a feasible schedule. Then, a move can consist of changing the starting times of a subset of jobs such that another feasible schedule is obtained. The collection of feasible solutions which can be constructed from the current one \mathbf{x} by a *single* move defines the *neighborhood* of \mathbf{x} and each of these solutions is called a *neighbor* of \mathbf{x} .

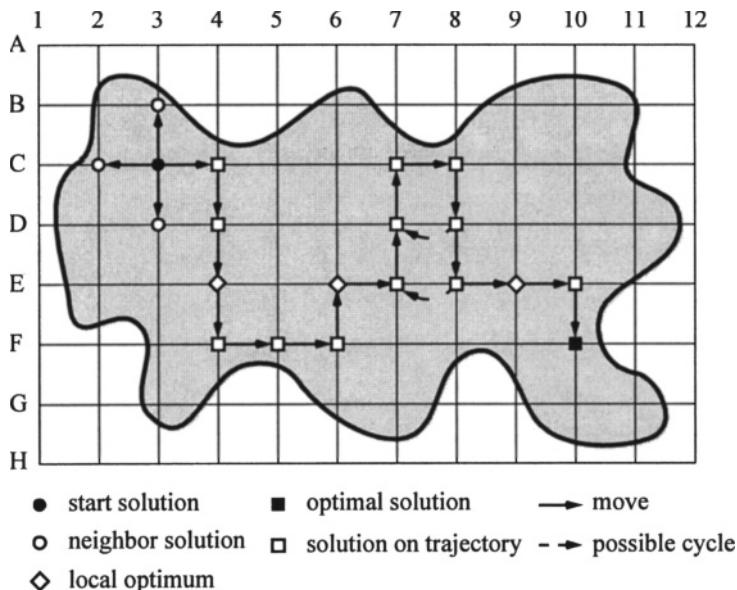


Figure 5.11. Solution space

When the search is guided by a *descent procedure* for a minimization problem only those moves are allowed which improve the current objective function value. The search ends when no improving neighbor solutions exist. Most often descent procedures are applied in a *steepest descent* version. Then, the entire neighborhood of \mathbf{x} is scanned in search of a neighbor \mathbf{x}' giving the smallest objective function value. Those solutions which have been visited while proceeding from the initial solution to the final one define the *search trajectory* which, figuratively speaking, corresponds to a path through the solution space.

Example. The circumstance is schematically depicted for a minimization problem in Figure 5.11. The solutions are represented by grid points with the feasible ones being contained in the shaded area. In order to proceed to a neighbor solution, we can move either vertically or horizontally along the grid by exactly one position. Starting with the feasible solution C3, the neighborhood is defined by the solutions B3, C2, D3, and C4 with only the last two ones having an objective function value smaller than C3. Following a steepest descent procedure, the move to solution C4 representing the best neighbor is performed. Subsequently, the solution D4 is visited before a local optimum is reached with solution E4. That is, all solutions in the neighborhood of E4 have a objective

function value which is not smaller than the current one. However, as indicated this solution is not optimal. An extended neighborhood could, e.g., be defined by moving in vertical and horizontal direction simultaneously.

In order to escape from local optimality, a *steepest descent / mildest ascent procedure* may be used. In each iteration, a move is realized which results in the largest improvement or the smallest deterioration of the objective function value. That is, as long as the neighborhood of the current solution contains improving solutions, this approach chooses the same moves as this is done by a steepest descent procedure. Then, a deteriorating move is performed in order to continue the search process. This process is repeated until a stopping criterion is fulfilled.

However, applying a steepest descent / mildest ascent procedure bears the difficulty that a locally optimal solution may be re-visited immediately after moving to a deteriorating solution or in subsequent iterations of the search process, respectively.

Example. In the solution space of Figure 5.11, performing the least deteriorating move possible from E4 may, e.g., lead to F4. If E4 is the neighbor of F4 with the smallest objective function value, the search process starts to cycle among these two solutions. Even if the procedure does not immediately return to the previous solution, the search may start *cycling*. Continuing the search may lead to the solution D7. Subsequently, the solutions C7, C8, and D8 are visited. Assuming that the solution D7 represents the best neighbor of solution D8, this solution will be re-visited and cycling between 4 solutions occurs.

5.3.1.2 Basic Principles of Tabu Search

In general, tabu search follows the steepest descent / mildest ascent approach. In order to avoid the problem of cycling between a number of solutions, tabu search prohibits certain moves which may lead to solutions examined previously by setting them *tabu active*. For this purpose, information on the search history has to be stored in some kind of *memory* and exploited later on in order to guide the search process.

To be more specific, a tabu search procedure starts with a feasible solution which is, e.g., computed by the application of a priority-rule based heuristic. Subsequently, the following steps are performed iteratively until a stopping criterion is fulfilled. In each iteration the best *admissible move* is computed. A

move is called admissible, if it is not tabu active or if some *aspiration criterion* is fulfilled (e.g., the move leads to a solution which improves the incumbent). Prior to performing a move, some *tabu management* is executed, i.e., information required for verifying the tabu status of future moves is stored in the memory. Each time, a solution is found improving the incumbent, this solution is stored as currently best one. This process is continued until some stopping criterion is fulfilled. According criteria may, e.g., refer to the number of iterations performed or to the computation time having emerged.

In the literature, a large number of different tabu search strategies have been proposed. In the main, they differ concerning the way of storing information for determining the tabu status of a move, i.e., whether it is tabu active or not. The two basic approaches for organizing the memory are briefly described in the following.

Recency Based Memory. This approach which is most commonly applied within tabu search procedures is based on storing information on moves which have been performed during the recent history of the search process. This follows the rationale that cycling can heuristically be avoided by excluding moves which lead to an inversion of the most recent ones. For this purpose, a *recency based memory* is introduced which manages the tabu information as follows.

In general, each move leading from a solution x to a solution x' can be characterized by one or more *attributes* which describe the operations performed to yield x' . Two attributes are called *complementary* when successively executing the corresponding moves results in the same solution as before. During the search process, the recency based memory keeps track of all attributes which are complementary to the ones describing moves performed in the recent past. Thus, these attributes represent moves which may lead back to previously visited solutions. Hence, a move is considered to be *tabu active*, when one or several of its attributes are stored within the recency based memory.

Most commonly, recency based memory is managed by creating one or several *tabu lists* which record the tabu active attributes. Each time when performing an admissible move, the attributes complementary to those describing the recent move are added to the lists where they remain for a certain number of iterations until the probability of re-visiting a solution due to these attributes is sufficiently small. The number of iterations for which an attribute is stored is referred to as *tabu tenure*. If moves are described by only a single attribute, this

value corresponds to the number of entries in the (single) tabu list, i.e., to its length.

Choosing the tabu tenure appropriately has considerable influence on the effectiveness of the tabu search procedure. If the value is too large, also moves may be considered to be tabu active which lead to yet unvisited solutions. By the way of contrast, if it is selected too small, cycling may occur. However, no rule can be given yielding an effective tabu tenure for all problem classes. In general, effective tabu tenures have empirically been shown to depend on the size of the problem instance. Furthermore, the tenure should not be fixed to a single value during the whole search process but should be varied dynamically. Basically, this can be done randomly from a range of values or systematically. In the latter case, information on the search history has to be exploited (cf. Section 5.3.1.3). Finally, the problem of determining an appropriate tabu tenure can also be attacked by using the *cancellation sequence method* (cf., e.g., Dammeyer et al. (1991)).

Example. Consider the solution space depicted in Figure 5.11. As stated above, a move consists of proceeding along the grid either vertically or horizontally. Therefore, moves can, e.g., be described by attributes giving the direction in which the search is continued. Then, the complementary attribute corresponds to the reverse direction. For example, when progressing from solution C3 to C4, the corresponding move can be described by the attribute "right". Accordingly, the complementary attribute is defined by "left". Assuming a tabu tenure of 3, the tabu list consists of the entries "up", "left", "left" when having reached the solution F6. Hence, the move to solution E6 with the attribute "up" is forbidden (tabu active) such that an improved solution may be missed. Furthermore, the trajectory leading to the optimal solution is interrupted. Reducing the tabu tenure to the value of 2, the search can follow the trajectory until solution D8 is obtained. At this point of the search, the tabu list contains the entries "left" and "up" and the move leading to the solution D7 can not be performed such that the search proceeds to solution E8. Due to the tabu list now having the entries "up" and "up", the move to solution E7 is admissible and cycling may occur if this solution represents the best neighbor.

Explicit Memory. Within this approach, a move is only prohibited if its execution really leads to a neighbor which has already been visited during the previous part of the search. However, storing and retrieving complete solutions generally requires an enormous amount of computer storage and time when

performed for each solution. To avoid this problem, the *reverse elimination method* can be applied. This method relies on accounting for logical relationships between the sequence of moves which have been performed throughout the search process so far (cf., e.g., Dammeyer and Voß (1993) and Voß (1995) for detailed descriptions). Furthermore, explicit memory may be emulated, e.g., by applying a hash code for describing a solution in order to reduce the storage and retrieval effort (cf., e.g., Woodruff and Zemel (1993), Carlton and Barnes (1996) as well as Section 5.3.2.2). Since for two solutions identical hash values may be computed leading to collision, sometimes moves may unnecessarily be considered to be tabu active.

Example. Due to storing all solutions having been visited so far, no move can be prohibited unnecessarily and no cycling can occur. Therefore, the search can follow the trajectory depicted in Figure 5.11, p. 192, until the optimal solution is obtained.

5.3.1.3 *Extensions of the Basic Approach*

In order to increase the effectiveness of the basic tabu search approach as described above, it may be extended by including one of the following components. For further possibilities and more comprehensive discussions see Glover and Laguna (1997).

Candidate List Strategy. Eventually, scanning the complete neighborhood of a solution may be computationally too expensive. This can be due to the neighborhood being large or the neighbors being too expensive to evaluate. Therefore, the examination of neighbors may be narrowed by applying a *candidate list strategy*. The most simple form of a candidate list strategy consists of randomly choosing k moves which are examined with k being a parameter of the search process. More evolved strategies can, e.g., be based on logical interrelationships (cf. Glover and Laguna (1997, section 3.2)). Another possibility of reducing the computational effort required for examining the neighborhood of a solution consists of computing bounds on the objective function values of the neighbors which can usually be performed more efficiently (cf., e.g., Baar et al. (1998) and Scholl et al. (1998)).

Varying Tabu Tenure. Choosing an appropriate tabu tenure has considerable influence on the effectiveness of guiding the search process such that solutions with a satisfying quality are obtained within the available computation time. In

order to avoid working with a disadvantageous value, the tabu tenure may be changed within a reasonable range after a certain number of iterations. This can either be done randomly or systematically. For example, tabu tenures which are too small can be detected by periodically repeated objective function values whereas values which are too large can be recognized by a continuous deterioration of solution quality over a number of iterations. A further possibility consists of storing the solutions already visited, e.g., by using hash codes, such that information on the history of the search process can be exploited to adjust the tabu tenure appropriately. The latter approach which is called *reactive tabu search* has been proposed by Battiti and Tecchiolli (1994 b) and is used within the tabu search procedure for GRCPSP described in the following section.

Aspiration Criteria. When applying a recency based memory, sometimes moves may unnecessarily be set tabu. In our above example with a tabu tenure of 3, this has, e.g., been true for the move from solution F6 to E6. Therefore, it may be advantageous to examine also those moves which are not admissible but possibly do not lead to a re-visit of a solution. Obviously, for all moves which immediately result in a solution better than the incumbent the active tabu status should be overridden (*global aspiration criterion*). Furthermore, it can be ignored for those moves resulting in an improved best objective function value known for solutions sharing particular values of one or more attributes, e.g., identical starting times of jobs in case of resource-constrained project scheduling (*local aspiration criterion*). Finally, an *aspiration-by-default* may be used when all available moves are classified to be tabu. In this case, the move which is "least" tabu is considered to be admissible. In the solution space depicted Figure 5.11, p. 192, such a situation may occur when, e.g., a move from solution F3 to solution G3 is performed. Since F3 represents the only neighbor of G3, the search process can only be continued when F3 is re-visited.

Intensification Strategies. Intensification strategies are applied to concentrate the search to promising regions of the solution space in order to examine them more thoroughly. For this purpose, some *frequency based memory* has to be introduced which provides information complementing the one of the recency based memory. Within this memory, *residence measures* are recorded for each attribute corresponding to the number of iterations where the attribute has possessed a certain value. Using this information, *residence frequencies* can be computed by dividing a residence measure by the number of iterations performed. A large residence frequency for a value indicates that it is attractive for

the respective attribute provided that the corresponding solutions are of high quality. Therefore, the attribute may be fixed to this value for a number of iterations in order to guide the search in the corresponding regions of the solution space. This is achieved by considering all moves to be tabu active which result in solutions for which the corresponding attribute yields a different value.

Diversification Strategies. Diversification strategies aim at driving the search into new regions of the solution space which have not been examined yet. This can be appropriate for two reasons. First of all, using a recency based memory, one can not guarantee that the search process does not start cycling. Secondly, it may be preferable to roughly examine different parts of the solution space instead of searching a single part thoroughly, in particular when the available computation time is limited. The following two strategies are most popular:

- The number of possible moves may be restricted by temporarily forbidding moves leading to solutions with attribute values which have frequently occurred during the search history for a number of iterations. For this purpose, the information provided by the frequency based memory can be exploited.
- Furthermore, the tabu search procedure may be restarted with a new feasible solution. This solution can either be constructed randomly or systematically. In the latter case, again information recorded by the frequency based memory may influence the construction process.

5.3.2 The Tabu Search Procedure RETAPS

In the sequel, we introduce a new tabu search procedure RETAPS (**R**eactive **T**abu **S**earch for **P**roject **S**cheduling) for solving GRCPSP and, hence, for RCPSP representing a special case of GRCPSP. First of all, the definition of the neighborhood is explained in Section 5.3.2.1. Furthermore, different candidate list strategies are defined. Finally, the tabu management performed and the diversification strategy employed are presented in Section 5.3.2.2.

5.3.2.1 *Definition of the Neighborhood*

When applying an improvement procedure, a suitable representation of solutions has to be determined which allows for an effective definition of moves. Within resource-constrained project scheduling, feasible solutions are com-

monly described by giving a schedule defining starting and finishing times for all jobs. However, this representation is not well suited for the use within improvement procedures, as can best be shown by means of our RCPSP example of Figure 5.3, p. 168 with $a_1 = 4$. Consider the schedule of Figure 5.12 to be the initial solution which has been obtained by applying SSS in combination with the SPT rule (cf. Figure 5.5, p. 170). Each move may consist in changing the scheduled starting time of a single job such that the resulting schedule is feasible again.

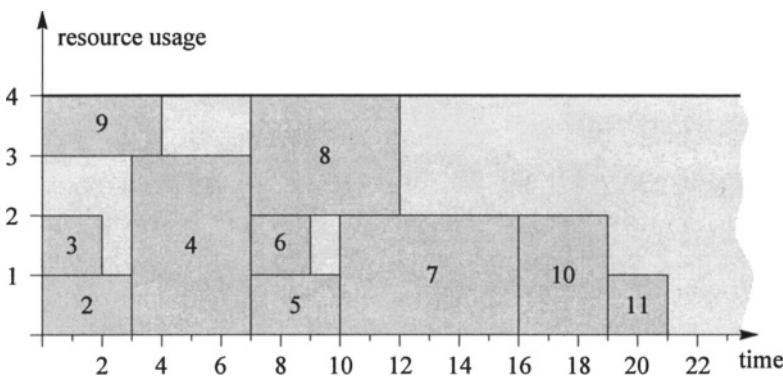


Figure 5.12. Initial solution

Then, the following 4 moves have to be performed in order to yield the optimal solution given in Figure 5.4, p. 169. The scheduled starting time of job 8 has to be increased to $SS_8 = 10$. This allows reducing the scheduled starting time of job 7 to $SS_7 = 7$ in the next iteration. Finally, job 10 and 11 can be left shifted to start at $SS_{10} = 13$ and $SS_{11} = 16$ in the last two moves. Though the optimal solution can be obtained by these moves, the example reveals the following disadvantages of operating on schedules:

- The resulting neighborhood is very large. For example, job 8 can be right shifted by 1, 2, 3, ... periods. This disadvantage may only partially be overcome by restricting to the examination of starting and finishing times of other jobs or of time points where fluctuations in the resource availability profile occur.
- For the initial solution, no move improving on the current objective function value exists. In fact, this value is not reduced before having performed the last move left shifting job 11. That is, within an improvement proce-

dure, the selection of promising moves is difficult. This problem becomes even more apparent, when the sequence in which incompatible pairs of jobs are executed has to be reversed. In order to yield a schedule in which job 8 is processed before job 4 starting from the one in Figure 5.12, a large number of non-improving moves has to be accepted (cf. Figure 5.6, p. 172).

Due to these reasons, a number of alternative solution representations have been developed for RCPSP which can be used within meta-heuristics and which can immediately be transferred to GRCPSP (cf., e.g., Kolisch and Hartmann (1999) for a survey). In the following, we present the one which has turned out to be best suited for the application within improvement procedures. The other representations have mainly been designed to be employed within genetic algorithms (cf. Section 5.3.3).

Representation of Solutions. The basic idea of this representation consists of coding a subset of all feasible solutions by relying on a scheduling scheme. Each active schedule can be generated by defining priority values appropriately and subsequently applying SSS. The same is true for each non-delay schedule using PSS. Instead of assigning priority values to jobs, a more direct representation is obtained by giving a *job list* L of the jobs ordered in the sequence in which they have to be scheduled. For the schedule given in Figure 5.12 and SSS, such an ordering is, e.g., given by $L = \langle 1, 3, 2, 9, 4, 8, 6, 5, 7, 10, 11, 12 \rangle$. Furthermore, the list may be constructed following a topological ordering, i.e., no job has a list position smaller than one of its predecessors (cf. Definition 3.1, p. 75). In this case, the jobs can be assigned a starting time without verifying whether all their predecessors have already been considered or not.

Remark 5.11. One major disadvantage of this representation deals with the circumstance, that a large number of different representations may result in the same schedule when being decoded using a scheduling scheme. For example, the jobs 3 and 2 may change positions in the list L given above with the application of SSS still resulting in the schedule of Figure 5.12. Furthermore, job 9 may even be put at the end of the list (in front of job 12) without yielding another solution. In order not to evaluate sequences yielding identical solutions, an appropriate candidate list strategy has to be introduced and careful tabu management has to be performed.

Within RETAPS, we use the following representation of solutions. In order to yield an optimal solution, only active schedules have to be examined, but it is

not sufficient to restrict to non-delay ones (cf. Remark 5.2, p. 166, and Remark 5.4, p. 167). Therefore, SSS is given preference to PSS. Furthermore, forward planning is applied because of considering GRCPSP. The sequence in which jobs have to be selected is defined by a topologically sorted job list. The same representation has, e.g., been used by Boctor (1996 b), Baar et al. (1998), Hartmann (1998), and Bouleimen and Lecocq (1998).

Definition of Moves. Having selected an appropriate representation of solutions, moves transforming one solution into another have to be described in order to define a neighborhood. Two basic operations can be performed on a given list (cf., e.g., Della Croce (1995)). A single job can be *shifted* to a lower or higher position in the list, respectively. Alternatively, a pair of jobs (h, j) not related by precedence may be *swapped*. In both cases, the operations have to be executed such that the topological ordering of the list is preserved which eventually requires changing the positions of respective predecessors and successors, too. Therefore, we describe the operations transforming the current job list $L = \langle 1, \dots, h, \dots, j, \dots, n \rangle$ into a modified one L' in more detail:

- Within a *forward shift*, the sequence of scheduling job h and job j is reversed by shifting job j in the list such that it is located immediately in front of job h . Predecessors of job j previously having a position in between the ones of job h and job j have to be moved, too. They are entered directly in front of job j such that their (relative) sequence is not changed.
- Accordingly, a *backward shift* which moves a job h such that it receives a position immediately after a job j can be defined. In this case, successors of job h formerly placed in front of job j take positions contiguously after job h according to their (relative) sequence in the previous list.
- By performing a *swap*, the jobs h and j change their positions in the list, i.e., job h is assigned the position of job j and vice versa. Subsequently, all successors of job h , previously located in front of j , are inserted immediately after h such that their former relative sequence is preserved. Furthermore, the affected predecessors of job j are positioned in front of job j following the same principle.

Example. As stated above, the list $L = \langle 1, 3, 2, 9, 4, 8, 6, 5, 7, 10, 11, 12 \rangle$ is a possible representation of the schedule given in Figure 5.12, p. 199. Job 10 may be forward shifted such that it is located immediately in front of job 8. However, in order to preserve the topological ordering of the list, this also requires

moving its predecessors 6, 5, and 7 thereby observing their relative sequence. The resulting list is $L' = \langle 1, 3, 2, 9, 4, 6, 5, 7, 10, 8, 11, 12 \rangle$. Subsequently performing a swap (2,8) yields $L'' = \langle 1, 3, 8, 9, 2, 4, 6, 5, 7, 10, 11, 12 \rangle$. Among the jobs previously located between 2 and 8, the jobs 4, 6, 5, 7, and 10 receive the positions directly after job 2 due to being successors of this job. By the way of contrast, job 9 is not affected.

Similar operations are, e.g., defined by Baar et al. (1998). Since both, a forward and a backward shift can be realized by a number of consecutive swaps, RETAPS restricts to evaluating swaps. That is, the neighborhood of a current schedule represented by a list L is defined by all schedules which can be obtained by a single swap move.

Candidate List Strategies. As described in the previous section, candidate list strategies are applied in order to reduce the effort required for evaluating the neighborhood in each iteration of the search process. Considering the neighborhood defined by the moves introduced above, this is necessary for two reasons. First of all, the number of pairs which can be considered for a swap is of the order $O(n^2)$. Secondly, to determine the improvement or deterioration associated with a move, a schedule has (partially) to be constructed which requires $O(m \cdot n^2)$ time.

Remark 5.12. The effort for evaluating a move may partially be reduced by preserving the scheduled starting times of those jobs preceding the first job h to be swapped in the list.

For this reason, Baar et al. (1998) propose to consider only jobs as possible candidates for a swap which are part of a critical sequence in the current schedule CS. For RCPSP, a *critical sequence* CSQ may be determined as follows. A graph $\bar{G} = (\bar{V}, \bar{A})$ is constructed with \bar{V} corresponding to the set of jobs J. An arc (h, j) is introduced for each pair of jobs for which the condition $SF_h = SS_j$ is true in CS. Each path connecting the dummy start job 1 and the dummy end job n in \bar{G} defines a critical sequence. In the schedule depicted in Figure 5.12, the only critical sequence is $CSQ = \langle 1, 2, 4, 5, 7, 10, 11, 12 \rangle$. Considering the list $L = \langle 1, 3, 2, 9, 4, 8, 6, 5, 7, 10, 11, 12 \rangle$ and following the proposal of Baar et al. (1998), only the swap move (5,7) is evaluated, because all other job pairs of CSQ are related by precedence. This results in a solution with a project completion time of 20. However, the swap move (8,7) is excluded, though it would result in a list representing the optimal solution given in Figure 5.4, p. 169. Fur-

thermore, for GRCPSP, a critical sequence may not exist at all depending on the current schedule. This is due to the fluctuations in the resource availabilities as well as to the minimum time lags. For these reasons, we have used the following candidate list strategies:

First of all, all pairs (h, j) are excluded from consideration which have identical starting times in the current schedule CS, i.e., $SS_h = SS_j$. For the schedule depicted in Figure 5.12 and the list $L = \langle 1, 3, 2, 9, 4, 8, 6, 5, 7, 10, 11, 12 \rangle$, this prevents from evaluating, e.g., the swap moves $(3, 2)$, $(3, 9)$, and $(2, 9)$ all resulting in lists L' for which the current schedule is obtained again. The same is true for the swap moves $(8, 6)$, $(8, 5)$, and $(6, 5)$.

Secondly, no pair (h, j) is considered for which job h is positioned in front of job j in L and $SS_j < SS_h$. In this case, the scheduled starting time of job j is obviously not influenced by the scheduling of job h . For example, consider the representation $L = \langle 1, 3, 2, 4, 8, 6, 5, 7, 10, 11, 9, 12 \rangle$ for the partial schedule depicted in Figure 5.12. Then, this situation occurs for the jobs 8 and 9. If a swap move is performed for these two jobs, the same schedule as before is obtained.

Besides these two strategies which are always applied within RETAPS the following additional candidate list strategies have alternatively been complemented by the way of trial:

- Among all pairs (h, j) representing possible candidates for a swap move in the current list L , only those are evaluated for which none of the jobs located in between represents a successor of job h or a predecessor of job j , respectively. In the example with $L = \langle 1, 3, 2, 9, 4, 8, 6, 5, 7, 10, 11, 12 \rangle$, this, e.g., excludes the swap move $(8, 10)$ due to the jobs 6, 5, and 7 being predecessors of job 10.
- Only those pairs of jobs are examined the execution of which partially overlaps in the current schedule without starting at the same time. For the schedule of Figure 5.12, this is true for the job pairs $(4, 9)$ and $(8, 7)$. However, in case of fluctuations in the resource availabilities, schedules may be obtained in which no job is active at all in some period intervals of the planning horizon due to the resource availabilities not being sufficient. As a consequence, for lists representing such schedules, no job being terminated before such an interval will ever be swapped with a job starting afterwards. In order to overcome this problem, additionally job pairs (h, j) where job h

finishes at the beginning of such an interval and job j starts at its end are considered for swaps.

- Basically, all pairs of jobs among which no precedence relationship exists are considered. However, only a certain number ns of the corresponding swap moves are evaluated which are randomly selected (with all moves having identical selection probabilities). Within RETAPS, this number is defined dynamically by $ns = \delta \cdot n$. If the last move performed has not led to a deterioration, we choose $\delta = 2$. Otherwise, $\delta = 1$ is used. The rationale behind setting the values of δ is the following. When no deterioration has occurred in the last move, further improvements of the current objective function value may be possible. Hence, it may be advantageous to examine the neighborhood more thoroughly.

Preliminary computational experiments have revealed the last strategy to be most efficient though representing the most simple one and including a random device. Therefore, it has been incorporated into RETAPS.

Remark 5.13. Due to GRCPSP being NP-hard in the strong sense, the evaluation of a job list may not result in a feasible schedule at all. That is, for single jobs the starting and finishing times may not observe the restrictions imposed by prespecified release and due dates. In order to consider such restrictions within the search process, violating a time window is penalized by adding the sum of all job durations to the completion time of the schedule. Furthermore, when a job can only be processed in a few period intervals due to fluctuations of the resource availabilities, it may not be assigned a starting time at all during the planning horizon in case of its predecessors being scheduled disadvantageously. Therefore, the planning horizon is virtually prolonged for each resource type assuming a per period availability which equals the maximum demand of any job.

5.3.2.2 Tabu Management and Diversification

As explained in Section 5.3.1.2, a recency based memory is usually incorporated within tabu search procedures. For this purpose, the moves defining the neighborhood are described by means of attributes. Unfortunately, applying this concept for the neighborhood introduced in the former section bears the following difficulty. Due to the moves manipulating job lists rather than schedules, recency based memory can only prevent from examining the same job list

twice. However, many permutations of the job list may represent one and the same schedule as discussed in Remark 5.11. That is, before accepting a deteriorating move and, hence, proceeding with the search process, all admissible moves leading to permuted job lists describing the current schedule will be performed. Furthermore, another problem consists of determining attributes that are sufficient in order to really avoid repetition of a certain job list (cf., e.g., Voß (1993, section 3.1.2.2) for an example).

Basic Tabu Management. To overcome the problems discussed above, *explicit memory* can be used within tabu search procedures. Since storing all solutions visited so far is usually prohibitive due to the computer storage being restricted, a *hashing function* can be applied in order to emulate an explicit memory by mapping a solution into a single integer *hash value* (cf., e.g., Woodruff and Zemel (1993) and Carlton and Barnes (1996)). Note that by employing a hashing function, moves may also be considered tabu which may lead to a yet unvisited solution. This is true, when a collision occurs, i.e., an identical hash value is computed for different solutions. Within RETAPS, a hash value $\varphi(\text{CS})$ is calculated for a schedule CS by the following hashing function:

$$\varphi(\text{CS}) = \left\lfloor \left(\sum_{j=1}^n z_j \cdot \text{SS}_j(\text{CS}) \right) / \Phi \right\rfloor \quad (5.1)$$

The values z_j are pseudo-randomly chosen integer numbers which are determined from an interval $[1, Z]$ each time the procedure is started. The value Φ is a sufficiently large prime number (cf. Remark 5.14 below). Computing $\varphi(\text{CS})$ for a schedule rather than for a job list has the advantage that different lists representing the same solution yield identical hash values such that inefficient moves can easily be detected. That is, a move is also considered tabu active if it results in a job list which is examined for the first time but corresponds to a solution that has already been visited.

Surprisingly, preliminary computational results revealed that incorporating such a strict type of memory influences the search process negatively though it effectively prevents the search from cycling. That is, it is disadvantageous to prohibit visiting a single or a number of solutions several times. Figuratively speaking this is always true, when the search trajectory guiding the search from one promising region of the solution space into another contains solutions already considered earlier. Then, it may be impossible to obtain solutions im-

proving the incumbent without allowing for re-visits. However, if this is permitted, it has to be guaranteed that the search does not start cycling, i.e., that from one of these solutions the search is continued into a new, yet unexplored direction. A simple possibility of realizing an according approach consists of considering a *recency based memory* containing only the last v solutions visited, corresponding to a tabu tenure of v . Then, a move is considered tabu active if it results in one of these solutions, whereas otherwise it is admissible. Whenever the resulting schedule has already been examined earlier, the tabu tenure is increased in order to reduce the probability of cycling (for a detailed discussion see below). Accordingly, it is decreased when no repetition has occurred for sufficiently large number of iterations.

Remark 5.14. Within our implementation of RETAPS, we used the values $Z = 2^{15} = 32768$ and $\Phi = 99991$. Hence, if the values of z_j are sufficiently random, the probability of a collision is approximately $1/99991 \approx 10^{-5}$.

Remark 5.15. The hash value computed is not stored explicitly. Instead an array with Φ entries is allocated when the procedure is started such that each entry can be addressed by the corresponding hash value. Within each entry, the *number of times* nv having visited the corresponding solution as well as the *number of the iteration* lv when the solution has been considered for the last time are recorded. Then, the tabu status of a move can be verified by simply comparing the sum of lv and the tabu tenure v to the number of the *current iteration* ir . In case of $lv + v < ir$, the move is admissible. Both informations can be stored in an integer variable with a size of 2 Bytes for each value, respectively, assuming that at most $2^{16} = 65536$ iterations are performed. That is, 4 Bytes are occupied by each entry. For the configuration described, this totals in a storage requirement of $99991 \cdot 4/1024 \approx 391$ KByte which is rather small in comparison to the available storage of current computer systems.

Varying Tabu Tenure. As indicated above, the tabu tenure is dynamically adapted depending on the history of the search process. This is achieved by following the principle of *reactive tabu search* (Battiti and Tecchiolli (1994 b) and Battiti (1996)) for which a number of successful applications have been reported recently (cf., e.g., Battiti and Tecchiolli (1994 a), Battiti and Tecchiolli (1995), Fink et al. (1998), Fink and Voß (1998), as well as Fink and Voß (1999)).

At the beginning, the procedure is started with a tabu tenure of $v = 1$. Each time, a solution is re-visited, the tabu tenure is increased as follows:

$$v := \min\{\max\{v + 1, \lfloor 1.2 \cdot v \rfloor\}, 2 \cdot n\} \quad (5.2)$$

The maximum value of the tabu tenure is limited to $2 \cdot n$ in order not to prohibit all available moves, though, in general, this does not represent a sufficient condition. However, this value has turned out to be a reasonable choice within our computational experiments. If still no admissible move exists, aspiration-by-default is applied. Furthermore, information on the development of the search process is recorded as follows. The length of the cycle causing the re-visit of the current solution can be computed by $ir - lv$ which is a value larger than v . The *moving average* ma of cycle lengths having occurred so far is calculated by using the formula $ma = 0.9 \cdot ma + 0.1 \cdot (ir - lv)$.

The tabu tenure is decreased again, when the number of iterations since re-visiting a solution for the last time exceeds the value ma . In this case, the distance measured in moves between the current solution and the last solution representing a repetition is larger than the average length of a cycle. Hence, probably no cycling can be caused by performing a single move. The adapted tabu tenure is calculated by:

$$v := \max\{\min\{v - 1, \lceil 0.8 \cdot v \rceil\}, 1\} \quad (5.3)$$

Diversification Strategy. The tabu management incorporated within RETAPS allows visiting a certain solution for several times as long as cycling is avoided. This follows the rationale that in order to examine the current region of the solution space thoroughly as well as to proceed into other promising regions, repetitions of solutions have to be accepted. However, when a subset of solutions is considered repeatedly, this may indicate that the region has either been evaluated nearly completely or that the search can not be prevented from cycling by further increasing the tabu tenure.

Therefore, a *diversifying move* is performed when a prespecified number of solutions have been visited for a certain number of times (cf. Battiti and Tecchiolli (1994 b)). Within RETAPS, this is done when 3 solutions have been visited twice. The diversifying move consists of restarting the procedure with computing a new feasible initial solution and setting the tabu tenure to $v = 1$. For this purpose, a topologically ordered job list is randomly generated and the according schedule is determined. In case it represents a schedule which has already

been examined earlier, the process is repeated. If after at most $\lceil n/2 \rceil$ trials still no yet unvisited solution has been obtained, the best trial one is selected as point of departure.

Besides this random approach, also a systematic one which is based on incorporating some frequency based memory has been examined. For each pair (h,j) of jobs not being related by precedence, the number of solutions visited is recorded for which job h has been positioned in front of job j in the according job list and vice versa. Then, a job list is constructed in which the order of those pairs (h,j) where h has frequently been located in front of j is reversed. However, computational experiments reveal this approach to be clearly outperformed by the random one. Therefore, it is not considered for being used in RETAPS.

5.3.3 Other Meta-Heuristic Based Procedures for RCPSP

In this section, we briefly discuss other meta-heuristic based approaches, using tabu search, simulated annealing, and genetic algorithms for solving RCPSP (cf., e.g., Kolisch and Hartmann (1999) for a survey). However, due to the large number of proposals differing considerably, we restrict to presenting the most successful ones for each type of meta-heuristic. Furthermore, references to other approaches are included. Modified *steepest descent procedures* are, e.g., proposed by Sampson and Weiss (1993), Leon and Balakrishnan (1995), and Naphade et al. (1997).

Tabu Search. Different tabu search procedures for RCPSP have been proposed by Lee and Kim (1996), Baar et al. (1998), Thomas and Salhi (1998), and Brucker and Knust (1999). In Baar et al. (1998), the two most promising approaches are compared to each other. The first one basically relies on the job list representation as well as on the neighborhood and the candidate list strategy based on critical sequences described in Section 5.3.2.1. However, the computational results presented by Baar et al. (1998) reveal that it is outperformed by the second one which can be outlined as follows. Note that due to the complexity of the approach the description is rather brief, giving only the basic idea.

Solutions are represented by a *schedule scheme* which describes a schedule by introducing either a conjunction, a disjunction, a parallelity relation or a flexibility relation for each pair of jobs (h,j) (cf. Section 6.4.4, pp. 258, for appropriate definitions). Different schedule schemes can be transformed into each

other by converting a flexibility relation into a disjunction or a parallelity relation and vice versa. For a given schedule scheme, a solution can be constructed by applying a special purpose heuristic which observes all conjunctions and disjunctions while concurrently trying to satisfy as much parallelity relations as possible. This process requires $O(m \cdot n^3)$ time.

Based on such a representation of solutions, the tabu search procedure can be described as follows. At the beginning, a promising initial solution is determined by applying some priority-rule based heuristics. Subsequently, a corresponding schedule scheme is constructed. In each iteration of the search process, the neighborhood is defined by all moves which transform a single flexibility relation into a parallelity one or vice versa. Furthermore, a candidate list strategy is used in order to restrict the number of admissible moves to be evaluated. For this purpose, the effect of each move is assessed by applying some lower bound arguments. Tabu management is performed by incorporating a recency based memory within which all pairs affected by a move in one of the last iterations are stored. The tabu tenure is dynamically varied within the range of 3 and 6. Finally, global aspiration is included.

Simulated Annealing. This meta-heuristic belongs to the class of improvement procedures and emulates the physical annealing process which consists of cooling a melted solid down to a low energy state (cf., e.g., Kirkpatrick et al. (1983), Aarts and Korst (1989), Dowsland (1993), and Domschke and Drexl (1998, section 6.5.1.3)). In each iteration, a possible move is selected randomly. If performing a move results in a solution with a better objective function value than the current one, it is immediately accepted and executed. Otherwise, it is realized with a certain probability which depends on the associated deterioration δ of the objective function value as well as on a parameter Θ called temperature. This probability is computed using a Boltzmann function of the type $e^{-\delta/\Theta}$. With the search process proceeding, the temperature is systematically reduced, e.g., after a prespecified number of iterations, such that the probability to perform a deteriorating move is lowered. The search is continued until some stopping criterion is fulfilled, e.g., the temperature falls below a certain level.

Simulated annealing based heuristics for RCPSP have been proposed by Boctor (1996 b), Lee and Kim (1996), Cho and Kim (1997), and Bouleimen and Lecocq (1998). Both, *Boctor (1996 b)* and *Bouleimen and Lecocq (1998)* use the job list representation introduced in Section 5.3.2.1 in combination with the serial scheduling scheme. An initial solution is constructed by employing the

SPT rule (cf. Table 5.2, p. 182). In each iteration, moves are obtained as follows. A job is selected randomly and is shifted to a random position in the list such that the topological ordering is preserved. To decide whether a move is accepted or not, the scheme described above is applied. Unfortunately, both references do not contain a detailed description of the cooling process, i.e., how the temperature Θ , representing the main parameter for controlling the search process, is reduced.

Genetic Algorithms. This meta-heuristic which has been inspired by the process of biological evolution is a member of the class of *evolutionary algorithms*, i.e., it manipulates a collection (*population*) of solutions (*individuals*) in each iteration. For comprehensive introductions as well as surveys on applications we refer to, e.g., Holland (1975), Goldberg (1989), Kolen and Pesch (1994), Michalewicz (1994), Dowsland (1996), Domschke (1997, section 1.3.2.3), and Reeves (1997).

Before starting a genetic algorithm, an initial population has to be determined, which consists of a prespecified number of feasible solutions. This can, e.g., be done by applying multi-pass priority-rule based heuristics. According to the improvement procedures described so far, each solution is encoded using some appropriate representation. In each of the following iterations, a new population of the same size is determined by performing the following operations. Executing *crossover operations* yields offspring solutions by mating. For this purpose, two solutions are randomly selected. Subsequently, their representations are combined into two new ones, e.g., by exchanging certain of their components (see below for an example) and the corresponding offspring solutions are constructed. Furthermore, following the principles of evolution, the individuals, i.e., their representations, can previously be modified randomly in order to obtain *mutations*. Finally, some measure of *fitness* (usually the objective function value) is applied to decide which parent and offspring solutions are selected to become a member of the next population. This process is repeated until some stopping criterion is fulfilled, e.g., a certain number of populations has been generated.

For RCPSP, genetic algorithms have been introduced by Leon and Balakrishnan (1995), Lee and Kim (1996), and Hartmann (1998). In Hartmann (1998) different representations of solutions are examined concerning their effectiveness including those incorporated in the previously presented approaches. These examinations reveal the representation relying on a job list and the serial

scheduling scheme to be most effective. Hence, *Hartmann (1998)* proposes the following genetic algorithm:

The initial population is computed by using a regret based biased random sampling method incorporating the LFT rule (cf. Section 5.2.4 and Table 5.2, p. 182). Subsequently, for each solution obtained, corresponding topologically ordered job lists are constructed. The crossover operation is basically implemented as follows. After selecting two job list L_1 and L_2 , a list position q is determined randomly. Then, the first part of the offspring list L'_1 is obtained by copying the first q positions from the existing lists L_1 . The second part is yielded by successively setting the jobs not yet considered onto the remaining positions according to the sequence in which they are contained in L_2 . Note that by this operation, the topological ordering of the lists is preserved. The second offspring list L'_2 is defined accordingly starting with the first q entries of L_2 . Finally, some mutation is performed by randomly changing positions of jobs with a certain probability such that the topological ordering is not violated. The new population is constructed by selecting those existing and offspring job lists for which the associated solutions possess the smallest project completion times, (i.e., the largest fitness values). Within the procedure of Hartmann (1998), a population size of 40 job lists (solutions) is used. In each iteration, the same number of offspring job lists (solutions) is additionally determined.

6 Exact Procedures

In this chapter, exact procedures for resource-constrained project scheduling are described. Though, in practice, using such procedures to optimally solve problem instances may often not be possible due to tight restrictions on the available computation time, their development can be justified for a number of reasons. In order to examine and compare the effectiveness of heuristic procedures, it is helpful to have optimal solutions on hand for the data sets considered. Furthermore, exact procedures may be used as heuristics themselves by simply stopping the branching process after a certain amount of time. Most importantly, for optimization problems being NP-hard in the strong sense like GRCPSP, their application provides the only possibility to verify whether a feasible solution for an instance exists or not.

For these reasons, a large number of *exact procedures* for solving resource-constrained project scheduling problems, in particular for RCPSP, have been developed. In general, the techniques applied within these procedures fall into one of the following basic categories which are commonly used in combinatorial optimization: integer programming, dynamic programming, or branch and bound methods. Since Patterson (1984) shows that the procedures using the latter technique clearly outperform the other ones, we only deal with such procedures. Integer programming based approaches are, e.g., described in Bowman (1959), Pritsker et al. (1969), Patterson and Huber (1974) as well as Patterson and Roth (1976) (cf. Section 3.2.1). A dynamic programming based method is proposed by Carruthers and Battersby (1966). For general surveys see, e.g., Herroelen (1972), Slowinski (1977), Patterson (1984), Herroelen et al. (1998), Brucker et al. (1999) as well as Section 6.4.

After briefly describing the major components of branch and bound in Section 6.1, a new branch and bound procedure called PROGRESS (**p**roject scheduling under **g**eeneralized precedence and **r**esource constraints) for solving GRCPSP is introduced in Section 6.2. Though the computational experiments in Section 7.5.1.1 reveal, that it represents the most efficient exact procedure currently on

hand for this problem, its performance can considerably be increased by integrating a new and general strategy for efficiently controlling the solution process which is presented in Section 6.3. The strategy relies on the observation that due to exploring the solution space very rigidly traditional branch and bound procedures may obtain good solutions as well as dominance information required for fathoming subproblems rather late during the solution process. This disadvantage can be overcome by *scattering* the search process. After decomposing the solution space into disjoint regions which can be examined independently the search process is controlled by diversification and intensification. Diversifying the search by intelligently "jumping" from region to region allows for collecting information on the structure of the solution space and, in particular, on regions the early examination of which is promising. Whenever such a promising subspace is identified, the search is intensified by comprehensively exploring this region in order to find improved solutions or to gather information being useful for fathoming. Based on this strategy, an improved version of PROGRESS, called SCATTER, is developed. Finally, the chapter is closed by a brief survey on existing procedures for RCPSP and GRCPSP in Section 6.4.

6.1 Components of Branch and Bound Procedures

The branch and bound method is a well known meta-strategy for exactly solving combinatorial optimization problems. For comprehensive descriptions, we refer to, e.g., Geoffrion and Marsten (1972), Johnson (1988), Hillier and Lieberman (1995, chapter 12), Roucairol (1996), Domschke (1997, section 1.2), and Scholl et al. (1997 b). Basically, it can be described as follows when assuming a minimization problem. The *branching process* starts with subdividing the initial problem into several subproblems. By continuously subdividing such subproblems again a multi-level enumeration tree with subproblems as nodes is constructed. Several basic principles help to avoid examining those subproblems which can not yield optimal solutions and, hence, can be fathomed. By *bounding*, subproblems for which a computed lower bound on the objective function value is at least as large as the one of the currently best known solution (i.e., the *incumbent*) are excluded. *Reduction rules* try to modify the problem data of a subproblem and possibly identify resulting contradictions allowing for fathoming the corresponding node of the tree. Finally, *dominance rules* rely on

detecting (dominated) subproblems whose examination can not provide a better solution than the best one obtainable from another (dominating) subproblem.

In the following, we give a brief description of the main components usually contained within branch and bound procedures assuming a minimization problem to be solved. Furthermore, we point out how the effectiveness of the bounding, reduction and dominance rules depends on the development of the solution process.

6.1.1 Branching Schemes

The branching scheme defines how the initial problem as well as the subproblems are subdivided (branched, developed) into further, usually disjoint, *subproblems*. As already mentioned, continuously subdividing such subproblems again leads to a multi-level *enumeration tree* with the subproblems corresponding to the nodes. The *root node* on the first level of this tree represents the initial problem. All nodes on subsequent levels are *descending nodes* which are obtained by developing an *ancestor node* of the immediately preceding level. The branching process stops, if for each undeveloped subproblem the optimal solution is either known or can efficiently be computed. The corresponding nodes are referred to as *leaf nodes*. This term is also used for nodes representing subproblems which can be fathomed, i.e., which need not be considered, because they can not yield a solution better than the incumbent or another node still to be developed. A path leading from the root node to any other node of the tree is called a *branch*.

The design of the branching scheme represents an important decision when creating a branch and bound procedure. Usually, a subproblem is developed by fixing one or more variables of the optimization problem to certain values or by restricting the possible ranges of their values. The branching scheme has to control which variables are selected and to which values they are set thereby influencing the size and the structure of the enumeration tree. In particular, this is true if fixing the values of some variables also allows for reducing the possible ranges of other variables. In this case, to keep the enumeration tree as small as possible, those variables should be selected first the fixing of which leads to the largest reductions in the possible ranges of the remaining ones. Therefore, usually variables for branching are selected heuristically, e.g., according to priority rules.

6.1.2 Search Strategies

Besides selecting a branching scheme, a search strategy defining the sequence in which the subproblems of the enumeration tree are considered has to be determined. Usually, this is done by orientating on lower bounds (LB) on the minimum objective function values of subproblems, which are primarily calculated for fathoming purposes (cf. Chapter 4 and Section 6.1.3). The lower bound of the root node defines a *global lower bound*, whereas those of all other nodes represent *local lower bounds*.

Most of the branch and bound procedures apply one of the two following basic search strategies or a combination of them (cf., e.g., Johnson (1988) and Scholl and Klein (1997)):

- Using a *depth-first search strategy (DFS)*, only a single branch of the tree is developed until reaching a leaf. Subsequently, the search continues with following the first alternative branch on the way back to the root node. That is, all descendants of a node are built before the search returns to its ancestor node(s). Usually, one of the following two variants of DFS is realized. If DFS is organized as a *laser search (DFSL)*, only one descending node is built and immediately developed in each node of the current branch. Alternatively, all descendants of a node may be constructed and sorted by increasing lower bound values. The one with the lowest lower bound value is developed first, while the others are stored in a *local candidate list*. At each revisit of the node, a descending node with the smallest local lower bound value is removed from the list and branched. This strategy is called *DFS with complete branching (DFSB)*.

A recent, hybrid strategy which aims at unifying DFSL and DFSB is the *local lower bound method (LLBM)* which has successfully been applied to simple assembly line balancing problems and the bin packing problem (cf. Klein and Scholl (1996), Scholl et. al (1997 a) as well as Scholl and Klein (1997, 1999 a, 1999 b)). Briefly, it can be described as follows (cf. Section 6.2.2 for a detailed discussion). Applying fast and simple bound arguments, the descending nodes are grouped into classes which are considered in non-decreasing order of the associated bound values. Due to restricting to simple bound arguments, the nodes of the current class can be identified efficiently when being generated. In contrast to DFSB, this allows for immediate branching without determining and storing all other descendants

before. Eventually, in order to avoid keeping a local candidate list, descending nodes may have to be generated several times depending on the problem considered.

- A *minimal-lower-bound search (MLB)* always selects and removes a not yet developed node with minimum bound value from a *global candidate list*. This node is completely branched by constructing all descending nodes which are stored in the list in non-decreasing bound order. At the beginning, the list contains only the root node. A variant of MLB is proposed in Demeulemeester and Herroelen (1997 b) which immediately develops one of the subproblems with the lowest bound among those created in the last branching step instead of taking the first one from the candidate list. The remaining subproblems are entered into the list as usual. This variant is referred to as MLB with immediate branching (**MLBI**).

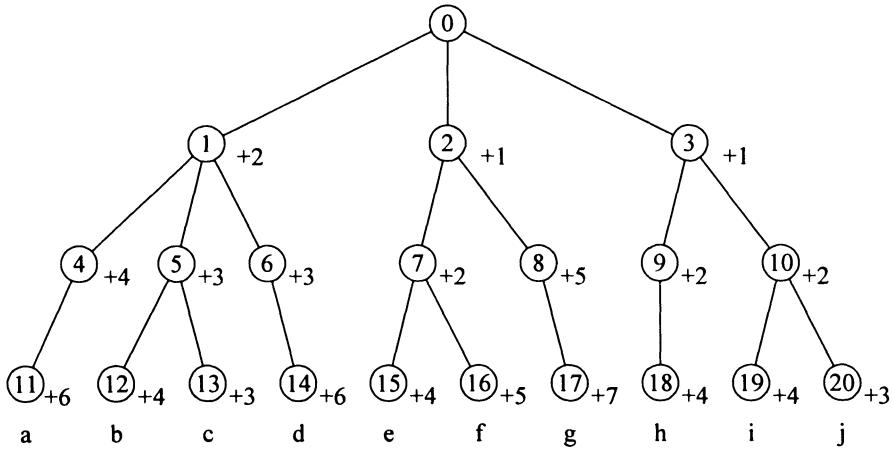


Figure 6.1. Enumeration Tree

Major advantages of DFSL are that feasible solutions are found very quickly and the number of nodes to be stored is low due to considering only a single branch at a time. Nevertheless, the search process of DFSL is somehow arbitrary, because no effort is spent for directing the search in promising parts of the tree, i.e., (near)-optimal solutions may be obtained rather late. To partially overcome this disadvantage of DFSL, the branching scheme of procedures incorporating this strategy is usually controlled heuristically. Alternatively, DFSB can be applied. However, determining all descending nodes in advance may be wasted effort, in case that one of the branches considered early leads to

a solution with an objective function value equal to the local lower bound of the node. Additionally, all nodes generated in ancestor nodes of the current branch have to be stored, which is sometimes impossible for complex scheduling problems even with well-equipped computer systems. LLBM overcomes these major disadvantages of DFSL by directing the search according to increasing local lower bound values but without completely generating (and storing) all subproblems. The need for storing *all* unbranched nodes created so far is also the main reason for the usual inapplicability of the different MLB strategies. Even for small problem instances the storage requirements exceed the available computer memory by far (cf., e.g., Demeulemeester and Herroelen (1997 b) and Nazareth et al. (1999)).

Example. The effects of choosing the different strategies can be shown by considering an example enumeration tree as given in Figure 6.1. Without the use of a search strategy, the branches (leafs) would be obtained from the left to the right due to the branching scheme. The amount by which the local lower bounds of the nodes exceed the global lower bound of the root node is denoted next to the nodes. Applying the different search strategies leads to the *visiting orders* in Table 6.1. For this simple example, LLBM generates the same tree as DFSB.

Table 6.1. Visiting order for different strategies

Strategy	Visiting Order
DFSL	0, 1, 4, 11, 5, 12, 13, 6, 14, 2, 7, 15, 16, 8, 17, 3, 9, 18, 10, 19, 20
DFSB (LLBM)	0, 2, 7, 15, 16, 8, 17, 3, 9, 18, 10, 20, 19, 1, 5, 13, 12, 6, 14, 4, 11
MLB	0, 2, 3, 1, 7, 9, 10, 5, 6, 13, 20, 4, 12, 15, 18, 19, 8, 16, 11, 14, 17
MLBI	0, 2, 7, 15, 3, 9, 18, 1, 5, 13, 10, 20, 6, 14, 4, 11, 12, 19, 8, 17, 16

6.1.3 Bounding Rules

Bounding rules are applied to reduce the size of the enumeration tree. A *local lower bound value* LB on the optimal objective function value is computed by relaxing the subproblem considered and solving the relaxation exactly (or solving the dual of the relaxation heuristically). This value is compared to the global upper bound UB which is the objective function value of the incumbent. If for a subproblem the value LB of a local lower bound is not smaller than UB, the node is fathomed, i.e., excluded from further consideration, because its so-

lution can not be better than the best one already known. Hence, the success of bounding in reducing the size of the enumeration tree depends on finding very good solutions early.

Usually, there is a trade-off between the effort required for computing lower bound values for fathoming a subproblem and the effort required for completely evaluating a subproblem by branching. Therefore, lower bound arguments should not rigidly be applied in each node of the enumeration tree. Instead, the decision whether to compute a local lower bound value or not should depend on the state of the solution process, e.g., on the number of variables already fixed and on the upper bound UB available. In general, the lower the level of a subproblem in the enumeration tree is, i.e., the less branching decisions have been made, the more computational effort can be saved by fathoming. That is, computationally rather expensive lower bound arguments should be computed only on low levels of the enumeration tree. Following this idea, time-consuming techniques for computing lower bound arguments are often applied only in the root node (cf. Section 4.1). This has the advantage that if the resulting global lower bound value obtained equals the optimal objective function value, the search process can stop as soon as a corresponding solution has been determined. However, if this is not the case, the bound computation has been wasted effort, because it does not help to fathom any of the subproblems.

6.1.4 Reduction Rules

The basic idea of reduction rules consists of modifying problem or subproblem data based on logical tests which exploit the structure of the problem considered. Typically, either the values of parameters are modified such that constraints become more restrictive or the range of values which can be assigned to a single or several variables is reduced.

In general, two basic types of reduction rules can be distinguished which differ concerning the use of an upper bound UB on the optimal objective function value. The first type which does not refer to UB leads to reduced problems possessing at least one optimal solution in common with the original one. These rules change parameter values and are commonly applied in order to increase the effectiveness of lower bound arguments. The second type requires a solution with UB to be known and, beyond the scope of the first type, tries to exclude all feasible solutions with an objective function value not smaller than

UB. Thus, if UB equals the optimal objective function value, no feasible solution for the reduced problem can exist. Therefore, this rule type may also allow for fathoming subproblems. If for a given value UB and a current subproblem, the reduction leads to a contradiction in the subproblem data, e.g., the range of possible values for a variable is empty, the subproblem need not be developed. As is the case with lower bound arguments, the success in fathoming subproblems depends on determining good upper bounds early. Note that the destructive improvement concept presented in Section 4.2 unifies reduction and bounding rules in order to obtain sharp lower bound arguments. In the literature, reduction rules are also referred to as immediate selection, edge finding, energetic reasoning, and interval consistency tests (cf., e.g., Dorndorf et al. (1999)).

6.1.5 Dominance Rules

By bounding and reduction rules, subproblems are fathomed if no descending node can lead to a feasible solution with an objective function value smaller than UB. By the way of contrast, dominance rules fathom (dominated) subproblems by showing that no descendant can lead to a feasible solution better than the best one obtainable by developing another (dominating) subproblem though this solution eventually has not been considered yet. That is, they can fathom subproblems irrespective of the current value of UB (cf., e.g., Ibaraki (1977)).

Two basic types of dominance rules can be distinguished. The first type (*transformation rules*) is based on transforming the best solution which can be found for the current subproblem into a feasible solution obtainable by another subproblem without increasing the objective function value. If this is possible, the current subproblem is dominated and can be fathomed, provided that the dominating subproblem has already been developed or is still to be examined. However, in the latter case, applying the dominance rule may lead to an increased effort due to obtaining the corresponding feasible solution with an (eventually) improved UB later such that in the meantime less fathoming is possible by bounding and reduction rules. This anomaly which has already been described in Kohler and Steiglitz (1974) can be avoided by applying such rules depending on the state of the search process (see, e.g., Klein and Scholl (1996)).

The second type of dominance rules (*storing rules*) is based on comparing attributes of the current subproblem to those of subproblems already developed. For this purpose, such attributes of non-dominated subproblems are stored. If the comparison of the attributes of a current and a stored subproblem shows that developing the current one can not improve on the best solution obtainable for the stored one, the current one is eliminated. Storing rules are usually more successful than transformation rules. However, their success depends on finding subproblems with dominating attributes early during the search.

6.2 The Branch and Bound Procedure PROGRESS

In this section, we describe the branch and bound procedure PROGRESS which has been developed for solving GRCPSP (cf. Klein and Scholl (1998 a)). The description is organized along the major components of branch and bound procedures introduced in the previous section. In order to ease the presentation, we use the example instance defined by the start-to-start project network given in Figure 6.2 and the resource availability profile depicted in Figure 6.3, p. 223. Since the project network corresponds to the one considered in Section 4.3, we refer to this section for an explanation of the notation. For a planning horizon of $\bar{T} = 30$, we assume a constant availability $a_{1t} = 3$ in the periods $t = 23, \dots, 30$. An optimal solution for this problem instance is presented in Figure 6.12, p. 239.

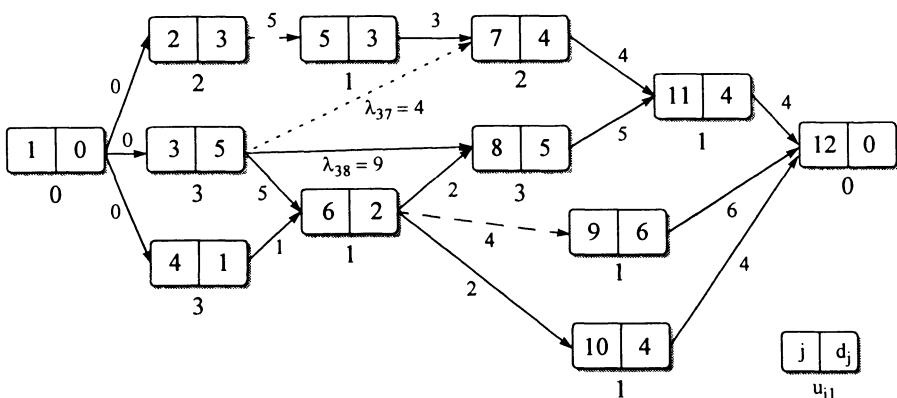


Figure 6.2. Project network

6.2.1 The Branching Scheme

In the following, we introduce the *parallel branching scheme* for GRCPSP which is closely related to the parallel scheduling scheme of priority rule-based heuristics (cf. Section 5.2.1.2) and to the approach proposed for RCPSP in Stinson et al. (1978). Further branching schemes which have been proposed for RCPSP in Demeulemeester and Herroelen (1992) and Sprecher (1997) and can be adapted by GRCPSP are described in Section 6.4.

Eligible Jobs. Each node of the enumeration tree represents a partial schedule PS and a *decision point* t (which terminates period t) such that all jobs $j \in J(PS)$ start before t ($SS_j < t$). Those jobs of $j \in J(PS)$ for which $SF_j > t$ are said to be *active*. For branching a node, the set $E(PS,t)$ of *eligible* jobs which can be assigned to the starting time t (and, hence, be started in period $t+1$), is determined (cf. also the Definitions 5.1-5.4, p. 163). For this purpose, a *modified forward pass* which results in earliest starting and finishing times $ES_j(PS)$ and $EF_j(PS)$ for all jobs $j \in J-J(PS)$ is performed. The decisions made in the ancestor nodes of the branch are considered by temporarily setting $rd_j := SS_j$ and $dd_j := SF_j$ for all jobs $j \in J(PS)$. Furthermore, since no unscheduled job is allowed to be started before the current decision point t , we set $rd_j := \max\{rd_j, t\}$ for all $j \in J-J(PS)$ by default. Those jobs $j \in J-J(PS)$ with $ES_j(PS) = t$ belong to $E(PS,t)$. Note that by performing the modified forward pass also a simple *critical path lower bound* $LBC1(PS) = EF_n(PS)$ is computed for the partial schedule PS (cf. Section 4.3.1.1).

Creating Subproblems. For branching, all subsets $E^* \subseteq E(PS,t)$ are systematically constructed which together with jobs still being active do not violate the resource constraints in period $t+1$ and the following ones. Each subset E^* defines a descending node (subproblem). To obtain the corresponding partial schedule PS' the current one PS is extended by setting $J(PS') = J(PS) \cup E^*$ and $SS_j = t$ for all $j \in E^*$. The appropriate next decision point t' is the smallest one with $t' > t$ for which a job $j \in J-J(PS')$ can be started according to the modified forward pass, an active job is terminated or at which the resource availability of at least one resource type increases.

Example. We consider the example instance given in the Figures 6.2 and 6.3. The partial schedule PS may contain the jobs $J(PS) = \{1, 2, 3, 4, 5\}$. At $t = 9$, the jobs 6 and 7 are eligible, i.e., $E(PS,t) = \{6, 7\}$. Hence, we obtain the subsets $\{\}$, $\{6\}$, $\{7\}$, and $\{6, 7\}$. After branching $E^* = \{6, 7\}$, the next time

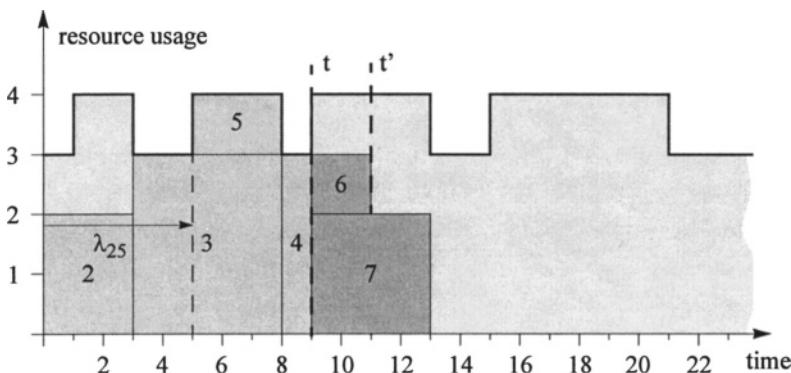


Figure 6.3. Partial schedule

point to be considered is $t' = 11$ at which job 6 is terminated and job 10 becomes eligible. Jobs 8 and 9 can not be scheduled due to their time lags to jobs 3 and 6, respectively. The first increase in the resource availability occurs at time point 15.

Constructing Subsets E^* . Within PROGRESS, the sequence in which the subsets $E^* \in E(PS, t)$ are generated is important. Therefore, we describe this process in more detail. The subsets $E^* \in E(PS, t)$ are systematically enumerated using a stack onto which the jobs are successively put according to increasing job numbers. This is continued until no further job $j \in E(PS, t)$ can additionally be scheduled at the time point t without the jobs on the stack together with the active ones causing a resource conflict. Afterwards, the job added the last, i.e., the one on the current top level of the stack, is removed and the next fitting one in $E(PS, t)$, having a larger number, is added. If no such job is found, the one on the previous level is replaced. Subsequently, the stack is refilled as long as possible. The remaining subsets are yielded by the same principle. The enumeration stops, when the job with the largest number is removed from the first level of the stack. A subset E^* is always obtained before removing a job from the stack. Furthermore, the empty set has to be considered.

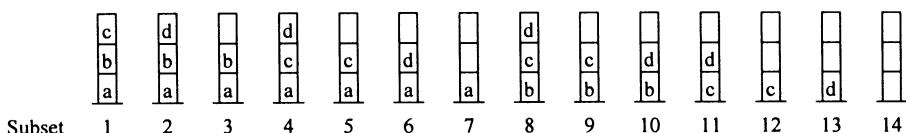


Figure 6.4. Building subsets using a stack

Example. Consider the set $E(PS, t) = \{a, b, c, d\}$ and assume that at most three jobs can be executed in parallel due to the resource constraints. Figure 6.4 shows the order in which the subsets are obtained. The process starts with adding the jobs a, b, and c. Since job d can not additionally be put onto the stack due to the resource constraints, the first subset to be branched is $E^* = \{a, b, c\}$. Subsequently, job c is removed and job d is added resulting in the second subset $E^* = \{a, b, d\}$. After thereby having examined all alternatives on the third level of the stack, job b on the second level is replaced by job c. Before this takes place, the subset $E^* = \{a, b\}$ is obtained. The process is continued until job d is removed from the first level of the stack. Finally, the empty set is yielded.

Remark 6.1. If $\lambda_{ij} = 0$ for a job $i \in E(PS, t)$ and one of its successors j , then job j may also belong to $E(PS, t)$. In this case, each feasible subset E^* , including job j , must also contain job i , which can easily be verified while constructing the subsets.

6.2.2 Local Lower Bound Method

The local lower bound method (LLBM) is a variant of the depth-first search strategy which combines the advantages of DFSL and DFSB by restricting to simple and fast lower bound arguments (cf. Section 6.1.2). In each node, an initial local lower bound value LLB is computed. For branching, the subproblems of the current node are subdivided into several classes according to their local lower bound values, with those having identical values being in the same class. The class considered first contains all subproblems with a local lower bound value equal to the initial LLB. Due to restricting to simple bound arguments, these subproblems can be identified efficiently when being generated. Each of them is immediately branched whereas all other subproblems are ignored for the time being. If branching all subproblems of the first class yields no solution with an objective function value LLB, the local lower bound LLB is increased to the next possible value of the subsequent class and the subproblems of this class are examined. If again no solution with an objective function value equal to the (new) current LLB is found, LLB is set to the next larger possible value. This process is repeated until all subproblems have been branched or the value of LLB is no longer smaller than the global upper bound UB.

LLBM overcomes the major disadvantage of DFSL by directing the search according to increasing local lower bound values but without accepting the disad-

vantage of completely generating (and storing) all subproblems and applying possibly expensive bound arguments as is the case with DFSB. However, as LLBM applies only fast bound arguments, the resulting sequences in which subproblems are branched might differ from those of DFSB.

For GRCPSP, subproblems can even be generated by LLBM in order of increasing lower bound values such that storing subproblems is obsolete. For other optimization problems, the generation in this order is not always possible. In this case, storing is avoided by applying the branching scheme several times and identifying the subproblems of the current class by their lower bound values. Nevertheless, if the number of classes is low, LLBM still outperforms other strategies due to the immediate branching of subproblems. This has, e.g., been shown for simple assembly line balancing problems in Klein and Scholl (1996) and Scholl and Klein (1997) as well as for the bin packing problem in Scholl et al. (1997) and is also valid for GRCPSP (cf. Section 7.5.1.2, pp. 309).

Applying LLBM to GRCPSP. In order to apply LLBM to GRCPSP, *start tails* $\psi_j = LF_n - LS_j$ are initially computed after a backward pass for each job j denoting the number of periods which have to pass between starting job j and completing the whole project. Note that these tails differ from the finish ones $\omega_j = LF_n - LF_j$ considered for computing lower bound arguments which refer to the number of periods between finishing a job and terminating the project, i.e., $\psi_j = \omega_j + d_j$ (cf. Section 4.1.1.4).

Then, for each node with a partial schedule PS and a decision point t , a simple initial local lower bound can be computed by $LLB = t + \max\{\psi_j \mid j \in E(PS, t)\}$, because none of the eligible jobs $j \in E(PS, t)$ can start before time point t and at least ψ_j periods have to pass between starting job j and finishing the project.

For the subproblems to be built, respective local lower bound values can easily be computed after calculating the *smallest possible next decision point* $t'_{\min} > t$. It corresponds to the minimum of the completion times of all active jobs, of the current earliest finishing times of all eligible jobs, and the next time point at which the availability of at least one resource type increases. Consider a subset E^* which is appended to the current partial schedule PS at decision point t . All jobs $j \in J - J(PS) - E^*$ can not be started before the next decision point t' for which t'_{\min} is a lower bound. Hence, a local lower bound LLB' of the corresponding subproblem can be computed as follows:

$$LLB' := \max\{LLB, t'_{\min} + \max\{\psi_j \mid j \in J - J(PS) - E^*\}\} \quad (6.1)$$

Within the parallel branching scheme, the subsets E^* can automatically be constructed in non-decreasing order of local lower bound values LLB' such that these values need not be computed explicitly. This is achieved by locally renumbering the jobs $j \in E(PS, t)$ according to non-increasing values of ψ_j and using this renumbering as the basis of constructing the subsets E^* . Whenever a job j with $t'_{\min} + \psi_j > LLB$ is removed from the top level of the stack, the local lower bound is increased to $LLB := t'_{\min} + \psi_j$ and a new class of subproblems is opened. This is due to the fact, that job j reentering the stack later requires removing a lower-numbered job h with a tail $\psi_h \geq \psi_j$. Hence, the local lower bound value LLB' of any subproblem still to be considered can not be smaller than the currently increased LLB .

Example. We consider an example with $E(PS, t) = \{a, b, c, d\}$ and $t = 10$. The job durations are $d_a = 6$, $d_b = 7$, $d_c = 5$ and $d_d = 7$. Furthermore, the jobs are already sorted according to non-increasing tails $\psi_a = 18$, $\psi_b = 16$, $\psi_c = 15$, $\psi_d = 14$. No job is active at time point $t = 10$ and at most three jobs can be executed in parallel due to the resource constraints. The first increase of the availability of one of the resources occurs at time point 16. Hence, the smallest possible decision point is $t'_{\min} = \min\{16, 10+6, 10+7, 10+5, 10+7\} = 15$.

We start with the initial local lower bound $LLB = 10 + 18 = 28$. Applying the branching scheme yields the first subset $E^* = \{a, b, c\}$ (cf. Figure 6.4, p. 223). Job d can not be added because otherwise the resource constraints are violated. Since $t'_{\min} + \psi_d = 15 + 14 = 29$ is larger than the current LLB , the first class of subproblems is empty. Hence, LLB is immediately increased to $LLB = 29$ and $E^* = \{a, b, c\}$ which is the first subproblem of the second class is branched. After removing job c from the stack, we obtain $LLB = t'_{\min} + \psi_c = 15 + 15 = 30$. For this value of LLB , the subsets $E^* = \{a, b, d\}$ and $E^* = \{a, b\}$ are branched. The further values of LLB and the subsets E^* contained in the corresponding classes are given below:

$$LLB = 31 : \{a, c, d\}; \{a, c\}; \{a, d\}; \{a\}$$

$$LLB = 33 : \{b, c, d\}; \{b, c\}; \{b, d\}; \{b\}; \{c, d\}; \{c\}; \{d\}, \{ \}$$

6.2.3 Bounding Rules

As already mentioned in Section 6.1.3, bounding is performed in order to reduce the size of the enumeration tree. For RCPSP, a large variety of lower bounds have been proposed (cf. Chapter 4). Two different concepts are most

successful in obtaining sharp (global) lower bounds: the destructive improvement technique introduced in Section 4.2.2 and the set covering based approach described in Section 4.1.2.3. Nevertheless, both concepts share the disadvantage that they are computationally rather expensive and can not be applied within branch and bound procedures except for the root node. For GRCPSP, only some simple bound arguments and the destructive improvement technique have been adapted so far (cf. Section 4.3).

Within our branch and bound procedure (including the root node), we restrict to using the capacity bound $LBC2'$ as well as the one-machine and the two-machine bound $LBM2'$ and $LBM3'$ the combination of which has been proved to be fast and effective (cf. Section 7.3.1, pp. 275). However, as computational results in Section 7.5.1.2, pp. 309, show, the effort for applying these additional bound arguments is not always justified due to the effectiveness of the bound calculation already contained in the local lower bound method. Since all three bound arguments used are described in Section 4.3.1, pp. 150, in detail, we only briefly describe how to modify them in order to obtain (local) lower bounds for a partial schedule.

Capacity Bound $LBC2'$. This argument discards all but one resource type r and determines the total remaining capacity requirement $\sum_{j \in J - J(PS)} u_{jr} \cdot d_j$ of all jobs which are not contained in the current partial schedule PS . This capacity requirement is contrasted to the residual availabilities which, considering the resource usage of the active jobs, are provided in the subsequent periods $t+1, t+2, \dots$, i.e., after the current decision point t . By doing so, a local lower bound $LBC2'(r)$ can be computed as the smallest time point for which the total residual capacity of the periods $t+1, \dots, LBC2'(r)$ exceeds or equals the total remaining capacity requirement. The maximum bound $LBC2'(r)$ obtained for all resource types $r = 1, \dots, m$ serves as a local lower bound value $LBC2'(PS)$ (cf. Section 4.3.1.1, pp. 152).

Machine Bounds $LBM2'$ and $LBM3'$. Those two arguments which have initially been proposed for RCPSP are based on identifying pairs and triplets of jobs which are incompatible, i.e., which can not be processed in parallel due to precedence or resource constraints (cf. Section 4.1.1.3, pp. 120). In Section 4.3.1.3, pp. 157, two basic possibilities are described how these bound arguments can be adapted to GRCPSP. The simple one consists of determining the

maximum resource availability $a_r^{\max} = \max\{a_{rt} \mid t=1, \dots, \bar{T}\}$ of each resource type $r = 1, \dots, m$ and any period and assuming this availability to be present during the complete planning horizon. Though this relaxation leads to a reduced bound quality, the corresponding bound values can be computed quite efficiently. The incompatibility of pairs and triplets can be verified once prior to starting the branching process. In each node of the branch and bound tree, the one-machine bound and the two-machine bound can be applied to the unscheduled jobs $J-J(PS)$ as described for RCPSP in Section 4.1.1.3 with the heads α_j being equal to the earliest starting times calculated for the current partial schedule PS. Then, the computation time required by the arguments is of the order $O(n^2)$ and $O(n^3)$, respectively.

The more complex variant relies on computing the number of periods for which each pair and triplet of jobs can be executed concurrently thus allowing to consider fluctuations in the resource availabilities (Section 4.3.1.3, pp. 157). Computing these values depending on the partial schedule needs $O(m \cdot n^2 \cdot \bar{T} + n^3)$ and $O(m \cdot n^3 \cdot \bar{T})$ time, respectively. Obviously, this computational effort is prohibitive when arising in each node of the branch and bound tree.

6.2.4 Reduction and Dominance Rules

In the sequel, we describe reduction and dominance rules for fathoming subproblems.

6.2.4.1 Core Time Rule

This rule is a reduction test which is based on the following idea. To improve on the current upper bound UB , a solution with a maximal objective function value of $UB-1$ has to be obtained. That is, according *latest starting times* $LS_j(UB-1)$ for $j = n-1, \dots, 1$ can be computed by performing a backward pass starting with $LS_n(UB-1) = UB-1$. Whenever, for a current partial schedule PS, the latest starting time $LS_j(UB-1)$ of a job j is smaller than its earliest finishing time $EF_j(PS)$, job j must be at least active during $[LS_j(UB-1), EF_j(PS)]$. This time interval is called *core time* of a job j (cf. Section 4.2.2.1, pp. 141, for a detailed discussion). The core time rule can now be described as follows.

Core Time Rule: Given a partial schedule PS with a decision point t . If for any period $q = t+1, \dots, UB-1$, the total resource demand caused by core times

in this period q exceeds the residual availability of any resource type $r = 1, \dots, m$, no feasible completion of PS with a project duration of $UB - 1$ can exist. Hence, the partial schedule PS need not be examined.

Example. Assume that the optimal solution for our example with a project duration of $UB = 22$ has already been obtained during the branching process (cf. Figure 6.12, p. 239). The current partial schedule PS may consist only of job 3 which has been scheduled to start at $t = 0$. The current decision point is $t = 5$. The earliest finishing times $EF_j(PS)$ and latest starting times $LS_j(UB - 1)$ of all jobs $j \in J - J(PS)$ are given in Table 6.2. Due to their core times, the jobs 7 and 8 have to be processed concurrently in period 14, i.e., between the time points 13 and 14. Their total resource demand is 5 units which exceeds the availability $a_{1,14} = 3$. Therefore, PS can not be completed to a schedule with a project completion time of $UB - 1 = 21$ and need not be considered.

Table 6.2. Calculation of core times

j	2	4	5	6	7	8	9	10	11	12
$LS_j(UB - 1)$	5	9	10	10	13	12	15	17	17	21
$EF_j(PS)$	8	6	13	8	17	14	16	12	21	21

6.2.4.2 Active Schedule Rules

In the following, we describe two transformation dominance rules which restrict the enumeration of solutions to active schedules. As it is well-known, this class of schedules contains at least one optimal solution for each GRCPSP instance (cf. Remark 5.2, p. 166).

In contrast to other approaches which explicitly check for a given partial schedule whether a scheduled job can either be locally or globally left shifted, we describe two rules which already avoid constructing such schedules, thus decreasing the computational effort required. Each rule considers a partial schedule PS with a decision point t as well as a set E^* to be branched resulting in a partial schedule PS' with decision point t' .

Semi-Active Schedule Rule: If there is an eligible job $j \in E(PS, t) - E^*$ which could additionally be started at time point t together with the jobs $h \in E^*$ without violating the resource constraints, it must not be scheduled at t' .

Otherwise, if job j is scheduled at time point t' , it will always be possible to perform a local left shift such that job j already starts at time point t . Hence, the resulting schedule can not be semi-active. Within PROGRESS, those jobs for which the semi-active schedule rule applies are excluded from $E(PS', t')$.

Example. In the example of Figure 6.5, job 9 can additionally be started at the decision point $t = 15$. Therefore, it must not be scheduled at $t' = 18$.

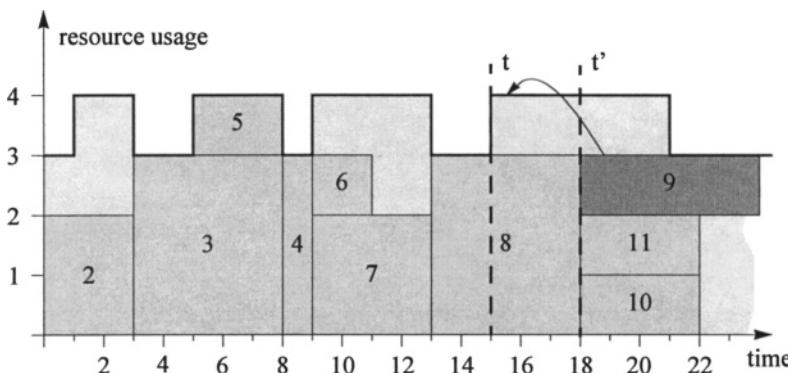


Figure 6.5. Example for the semi-active schedule rule

For the active schedule rule, we have to store the *earliest possible starting time* (decision point) EPS_j of each unscheduled job, since which it could be executed in addition to the jobs of $J(PS)$ without being preempted or violating precedence or resource constraints. The values EPS_j can be computed without additional effort while constructing the branch leading to the current node.

Active Schedule Rule: If there exists an eligible job $j \in E(PS, t) - E^*$ with $EPS_j + d_j \leq t'$, E^* need not be branched.

Otherwise, after branching E^* , job j can always be globally left shifted by assigning the starting time $SS_j = EPS_j$, because it will be finished until time point t' the latest and, hence, can not lead to resource conflicts together with those jobs started at time point t' or later.

Example. Let the partial schedule PS with $J(PS) = \{1, 2, 3, 4, 5, 6\}$ and $t = 11$ in Figure 6.6 be given. The jobs 7 and 10 are eligible. Job 7 can already be started at time point 9 together with job 6, such that $EPS_7 = 9$. Branching only $E^* = \{10\}$ results in a decision point $t' = 13$ at which job 9 becomes eligible. Hence, PS' with $J(PS') = \{1, 2, 3, 4, 5, 6, 10\}$ can not be completed to an active

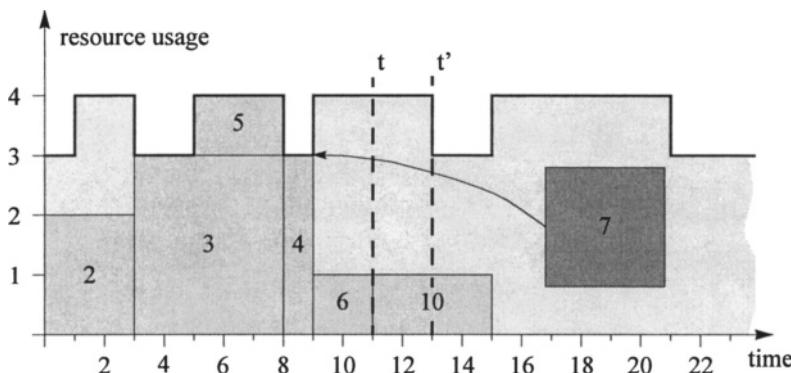


Figure 6.6. Example for the active schedule rule

schedule, because it will always be possible to globally left shift job 7, such that it starts at time point 9 and finishes at time point 13.

6.2.4.3 Supersession Rule

This transformation dominance rule excludes partial schedules by comparing scheduled to unscheduled jobs. It represents an adaptation of similar rules which have been proposed for the simple assembly line balancing problem (cf., e.g., Jackson (1956) and Scholl and Klein (1997)). The rule is based on the potential dominance between jobs. A job j potentially dominates a job h if the following conditions are true:

- $d_j \geq d_h$
- $u_{jr} \geq u_{hr}$ for $r = 1, \dots, m$
- $F_j \supseteq F_h$ with $\lambda_{ji} \geq \lambda_{hi}$ for all $i \in F_h$

In case that all conditions are fulfilled as equations, the lower-numbered job is defined to be the potentially dominating one.

Supersession Rule: A current partial schedule PS is excluded from branching, if a job $h \in J(PS)$ can be replaced by a job $j \in J - J(PS)$ potentially dominating h , such that j finishes until the current decision point t without violating the precedence or resource constraints.

The rationale behind the supersession rule is the following. For the current partial schedule PS, the potentially dominating job j has to be scheduled at time point t or later. However, job j is "harder" to schedule than the dominated one h ,

because its execution requires at least the same time and resources. Furthermore, each successor i of job h can not become available before job j has been scheduled and has to wait $\lambda_{ji} \geq \lambda_{hi}$ periods before it becomes eligible. Hence, the dominated schedule PS (containing job h) can not lead to a solution with a smaller project completion time than the dominating one (containing job j instead of h), for which the potentially dominated job h would have to be started at time point t or later instead of job j .

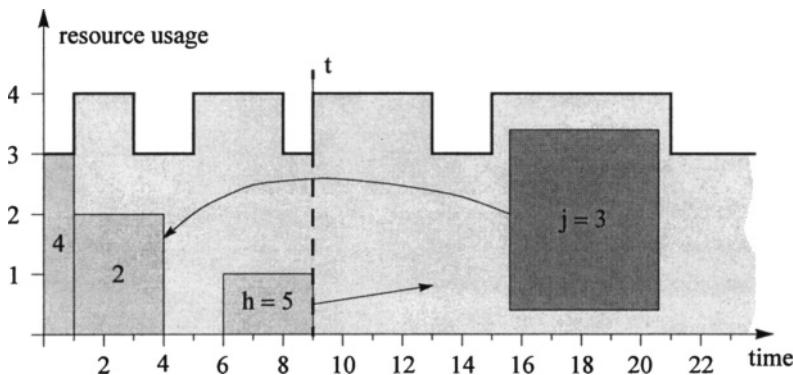


Figure 6.7. Example for supersession rule

Example. Consider our initial example for which job 3 potentially dominates job 5 (cf. Figure 6.2). For the partial schedule in Figure 6.7, job 5 can not be started earlier than time point 6 due to $\lambda_{25} = 5$. At the current decision point $t = 9$, job 3 can replace job 5 starting at time point 4 and finishing at time point 9 and, hence, the given partial schedule is dominated. For the dominating partial schedule, job 5 has to be scheduled at time point $t = 9$ or later instead of job 3. The only successor of job 5 (job 7) is also successor of job 3 and, hence, will not become eligible before both job 3 and job 5 have been scheduled. Since $\lambda_{37} > \lambda_{57}$, executing job 3 before job 5 results in a smaller earliest starting time for job 7 than vice versa. Additionally, the jobs 6 and 8, which are only successors of job 3 become eligible earlier.

6.2.4.4 Schedule storing rules

For RCPSP, it has been shown that the most efficient dominance rules rely on comparing the current partial schedule with previously evaluated, non-dominated ones which are stored for this purpose (cf., e.g., Demeulemeester and Herroelen (1997 b) and Sprecher (1997)). If the comparison shows that the current

partial schedule \bar{PS} with decision point t can not lead to a better solution than a previously stored one \bar{PS} with decision point \bar{t} , the current partial schedule is dominated and need not be examined. The following conditions are sufficient for this dominance:

- In both partial schedules \bar{PS} and PS, the same set of jobs is scheduled, i.e., $J(\bar{PS}) = J(PS)$.
- The decision point \bar{t} of the stored partial schedule is not larger than the decision point t of the current one, i.e., $\bar{t} \leq t$.
- The earliest starting times of all unscheduled jobs $j \in J - J(PS)$ fulfill the condition $ES_j(\bar{PS}) \leq ES_j(PS)$.
- In each period $q = t+1, t+2, \dots$, the jobs of \bar{PS} still active do not require more capacity of the resources types $r = 1, \dots, m$ than the active jobs of schedule PS.

Example. Consider the stored partial schedule \bar{PS} and the current one PS given in Figure 6.8. The same set of jobs is scheduled and the decision points \bar{t} and t are identical which is also true for the earliest starting times of the unscheduled jobs: $ES_8(PS) = 13$, $ES_9(PS) = 13$, $ES_{11}(PS) = 18$. Since for both partial schedules (only) job 10 is active until time point 15, the last condition is fulfilled, too. Therefore, evaluating PS can not lead to a better solution than the best one obtained earlier when examining all possible extensions of \bar{PS} .

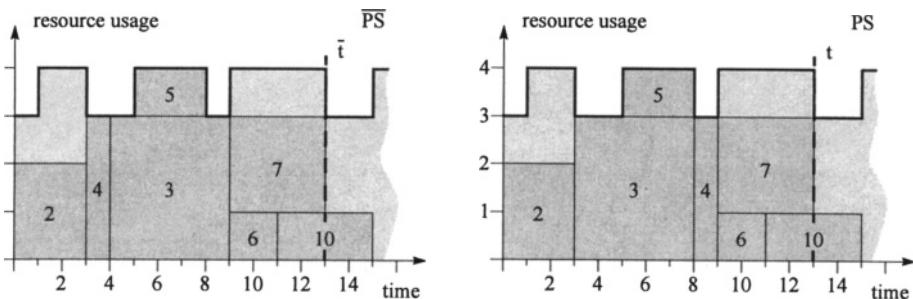


Figure 6.8. Example for schedule storing rule

Unfortunately, checking for dominance relationships between partial schedules as described above is computationally expensive and requires very much computer memory for storing the relevant data. Therefore, several concepts which avoid this effort by confining themselves to detecting only a subset of all possi-

ble dominance relationships have been proposed for RCPSP (cf., e.g., Stinson et al. (1978), Demeulemeester and Herroelen (1992), and Sprecher (1997)). However, those concepts can not immediately be transferred to GRCPSP due to the latter considering minimum time lags. For example, in case of RCPSP, the starting times of jobs terminated until the current decision point t are of no relevance, whereas for GRCPSP they may determine when unscheduled jobs become eligible. Hence, Demeulemeester and Herroelen have adapted their RCPSP cutset dominance rule to GRCPSP (cf. Demeulemeester and Herroelen (1997 a)). The major restriction of this adaptation is that no change in the availability of any resource type is allowed to occur between the current decision points of two partial schedules which are compared to detect a dominance relationship. Owing to this severe restriction, we describe a new version of such a dominance rule.

When using the parallel branching scheme, the decision point t of a partial schedule PS equals the smallest earliest starting time of all unscheduled jobs $j \in J - J(PS)$. Hence, for a stored partial schedule \bar{PS} and a current one PS with $J(\bar{PS}) = J(PS)$, \bar{t} can not be larger than t , if $ES_j(\bar{PS}) \leq ES_j(PS)$ for all $j \in J - J(PS)$. That is, the decision points of the partial schedules need not be compared explicitly. Also, the last condition considering the resource utilizations is checked only implicitly. Obviously, all scheduled jobs $j \in J(\bar{PS})$ with $SF_j(\bar{PS}) \leq t$ need not be considered, because they do not use resources in the periods $q = t+1, t+2, \dots$. For the remaining jobs $j \in J(\bar{PS})$ which are still active at time point t , $SF_j(\bar{PS}) \leq SF_j(PS)$ is a sufficient condition to guarantee that the resource utilization of \bar{PS} does not exceed the one of PS in any period q and for any resource type r (cf. Demeulemeester and Herroelen (1992)). As a consequence, for each non-dominated partial schedule, only the following data has to be stored: The earliest starting times of all unscheduled jobs, the set of jobs being active as well as their scheduled finishing times.

In order to reduce the memory requirements and to facilitate checking the dominance conditions, these data are represented by using *latest feasible starting times* $LFS_j(PS)$ of *scheduled* jobs. That is, we delay the scheduled jobs as far as possible such that the unscheduled ones can realize their earliest starting times $ES_j(PS)$ and the resource demands in the periods $q = t+1, t+2, \dots$ are not increased. These latest feasible starting times $LFS_j(PS)$ (for $j \in J(PS)$) are obtained by a modified backward pass where unscheduled jobs $j \in J - J(PS)$ are assumed to start at their earliest starting times $ES_j(PS)$, i.e.,

$LFS_j(PS) = ES_j(PS)$. Jobs $j \in J(PS)$ scheduled to finish at time $SF_j(PS)$ are not allowed to terminate later than $\max\{SF_j(PS), t\}$. With the latest starting times $LFS_j(PS)$, the simple schedule storing rule can be formulated as follows:

Simple Schedule Storing Rule: A current partial schedule PS is dominated by a stored one \bar{PS} , if $J(PS) = J(\bar{PS})$ and $LFS_j(PS) \leq LFS_j(\bar{PS})$ for all $j \in J(PS)$.

Note that the simple schedule storing rule considers only partial schedules containing the same set of jobs. This corresponds to the proposals made for RCPSP in the literature so far. However, if it is possible to schedule the same set of jobs more efficiently, it might also be possible to terminate additional jobs (within \bar{PS}) until the decision point t of the current partial schedule PS . Following this idea, we introduce the extended schedule storing rule which also compares the current partial schedule PS to stored ones consisting of a superset of jobs.

Extended Schedule Storing Rule. A partial schedule PS with the decision point t is dominated by a previously stored one \bar{PS} with $J(\bar{PS}) \supset J(PS)$, if $LFS_j(\bar{PS}) \leq LFS_j(PS)$ for all jobs $j \in J(PS)$ and $LFS_j(\bar{PS}) + d_j \leq t$ for all jobs $j \in J(\bar{PS}) - J(PS)$.

The latter condition ensures that jobs additionally scheduled in \bar{PS} neither increase the resource usage in the periods $q = t+1, t+2, \dots$ nor delay any of the unscheduled jobs.

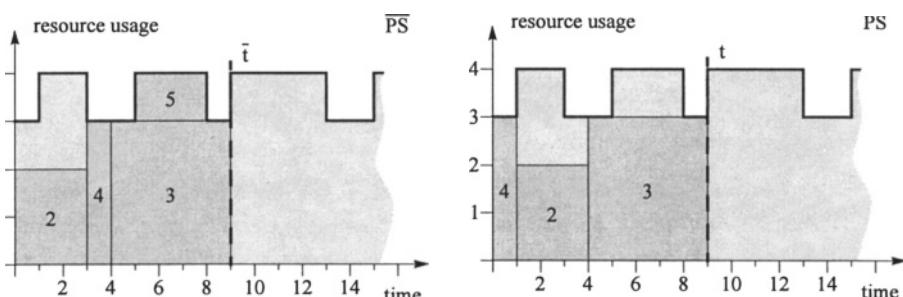


Figure 6.9. Example for extended schedule storing rule

Example. Figure 6.9 shows an example considering two partial schedules PS and \bar{PS} with decision points $t = \bar{t} = 9$. $J(\bar{PS})$ additionally contains job 5 such that job 7 is already eligible at $\bar{t} = 9$ unlike the case of PS . Furthermore, job 5 does not occupy resources beyond the decision point $t = 9$ of PS such that PS is dominated.

Storing Partial Schedules. The latest feasible starting times $LFS_j(PS)$ of each non-dominated partial schedule PS are stored in an array with an entry for each job $j = 1, \dots, n$. To identify the unscheduled jobs, we set $LFS_j(PS) = \infty$ for $j \in J - J(PS)$. All arrays obtained for partial schedules containing the same set of jobs are subsequently stored in a single-linked list. Within the list, the elements (arrays) are ordered lexicographically such that one can stop searching for dominating schedules when $LFS_j(\bar{PS}) > LFS_j(PS)$ occurs the first time. The different lists are addressed by an open hashing technique (in this context see Section 5.3.2.2, pp. 204).

For the extended schedule storing rule, all possible supersets of $J(PS)$ have to be enumerated in order to find the stored partial schedules \bar{PS} possibly dominating PS. However, this is prohibitive. Therefore, the maximum number ζ of jobs by which $J(\bar{PS})$ differs from $J(PS)$ is limited.

Storing all partial schedules considered so far may use a large amount of computer memory and increase the effort of retrieving the latest feasible starting times of partial schedules. Therefore, whenever a non-dominated partial schedule PS is stored, the list in which it is entered is scanned and all partial schedules dominated by PS are deleted. This can easily be done comparing latest feasible starting times.

6.2.5 Example

Figure 6.10 shows the enumeration tree which is obtained by PROGRESS when solving the example defined by the project network of Figure 6.2, p. 221, and the resource profile defined in Figure 6.3, p. 223. In order not to overload the presentation, among the bounding, reduction and dominance rules only the critical path bound, the core time rule and the active schedule rules are applied. The nodes are numbered according to the visiting order (first entry of each node). The second entry denotes the decision point t associated with each node. The subsets E^* used for branching are given next to the arcs. For a planning horizon of $\bar{T} = 30$, we assume a constant availability $a_{1t} = 3$ in the periods $t = 23, \dots, 30$. The start tails ψ_j of the jobs $j = 2, \dots, 11$ are given in Table 6.3.

Table 6.3. Start tails of non-dummy jobs

j	2	3	4	5	6	7	8	9	10	11
ψ_j	16	18	12	11	11	8	9	6	4	4

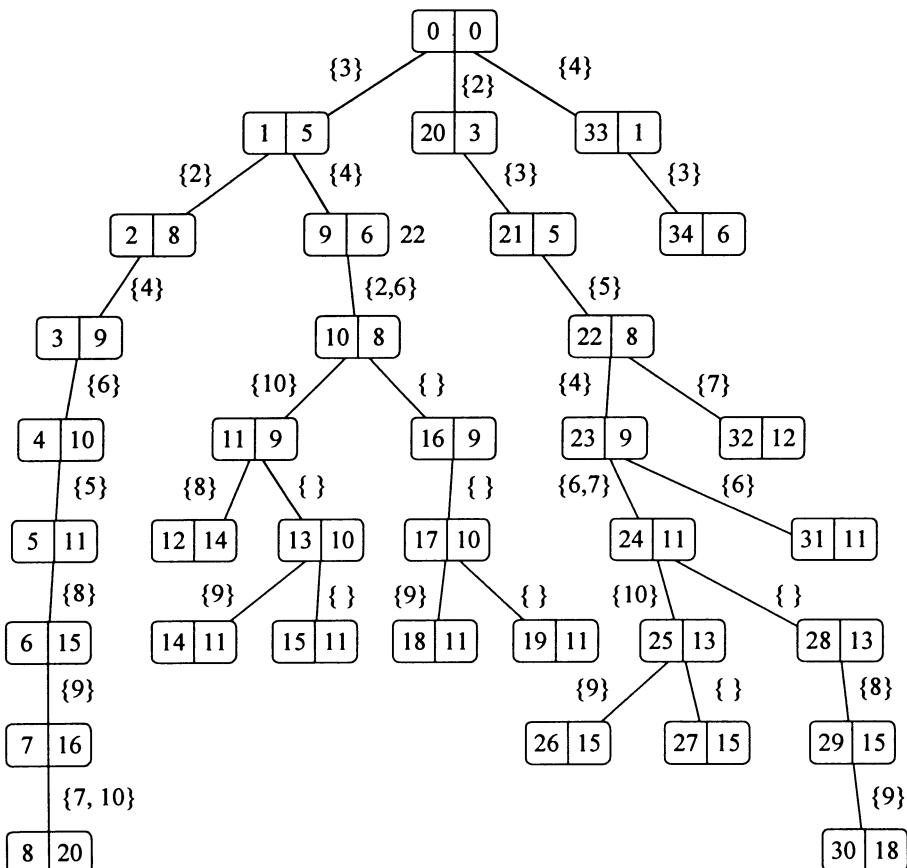


Figure 6.10. Enumeration tree of PROGRESS

In the root node 0 with decision point $t = 0$, we obtain the set of eligible jobs $E(PS, t) = \{2, 3, 4\}$. Since, due to the resource constraints, no pair of these jobs can be scheduled in parallel, the subsets E^* which have to be considered for developing subproblems are $\{2\}$, $\{3\}$, $\{4\}$, and $\{ \}$. Following the renumbering $(3, 2, 4, 5, 6, 8, 7, 9, 10, 11)$, the set $E^* = \{3\}$ is branched first. Subsequently, the jobs 2 and 4 are assigned the scheduled starting times $SS_2 = 5$ and $SS_4 = 8$ in the nodes 1 and 2, respectively. In node 3 with $t = 9$, only job 6 is eligible and, hence, scheduled. The same is true for job 5 in node 4 at $t = 10$. Note that it can not be started earlier due to the minimum time lag $\lambda_{25} = 5$. Having started job 8 at time point 11 in node 5 (instead of job 10 due to the renumbering), no idle capacity remains in the periods 14 and 15. Therefore, the next decision point is $t = 15$ at which either job 9 or job 10 can be scheduled with job 9 being

chosen because of its larger start tail. After having completed job 8 at $t = 16$ (node 7), the jobs 7 and 10 can be scheduled in parallel to job 9 which is still active. Finally, in the leaf of the branch, job 11 is the only remaining job such that no branching is necessary for obtaining a solution with $UB = 24$ (cf. Figure 6.11).

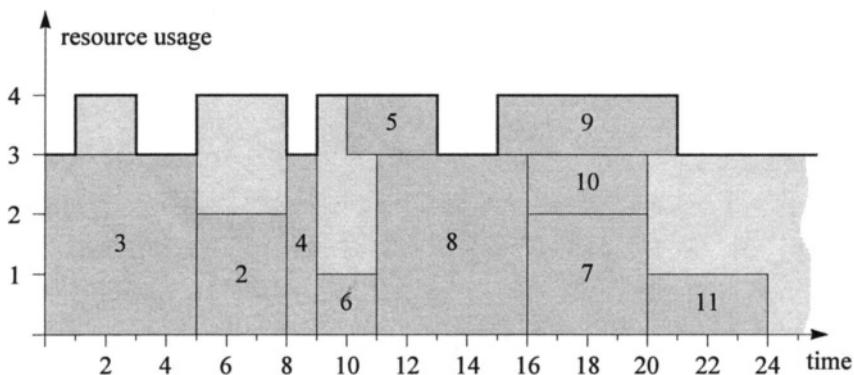


Figure 6.11. Solution of node 8

Subsequently, the procedure can trace back to node 1 without considering alternative branches by applying the core time rule in the nodes 7 to 2. For example in node 2, the current partial schedule PS consists of the jobs 3 and 2 with $SS_3 = 0$ and $SS_2 = 5$. The current decision point is $t = 8$. The earliest finishing times $EF_j(PS)$ and latest starting times $LS_j(UB-1)$ of all jobs $j \in J - J(PS)$ are given in Table 6.4. Due to their core times, the jobs 7 and 8 have to be processed concurrently at least in period 16, i.e., between the time points 15 and 16. Since their total resource demand exceeds the availability $a_{1;16} = 4$, PS may not be completed to a schedule with a project completion time of at most $UB - 1 = 23$ and need not be considered.

Table 6.4. Calculation of core times for $UB = 24$ in node 2

j	4	5	6	7	8	9	10	11
$LS_j(UB-1)$	11	12	12	15	14	17	19	19
$EF_j(PS)$	9	13	11	17	16	19	15	21

Developing the first alternative branch from node 1 finally leads to node 12. Its decision point $t = 14$ is obtained because of job 8 absorbing the resources completely in period 14. The node can be fathomed by calculating the initial local

lower bound value $LLB = 14 + \psi_5 = 25 \geq UB$. In the nodes 14, 15, 18 and 19, the core time rule applies in a similar manner as before. In node 16, the branching of $E^* = \{8\}$ with the next decision point $t = 14$ would not result in an active schedule because job 10 could always be assigned the starting time 8. The same is true after returning to the node 9 for the subproblem which would be obtained by the set $E^* = \{2\}$ due to job 6. After removing job 2 from the stack in order to yield $E^* = \{6\}$, the node 9 can be fathomed by bounding. The next potential decision point is $t'_{min} = 8$ (termination of job 6). Hence, the local lower bound can be increased to $LLB' = t'_{min} + \psi_2 = 8 + 16 = 24 \geq UB$.

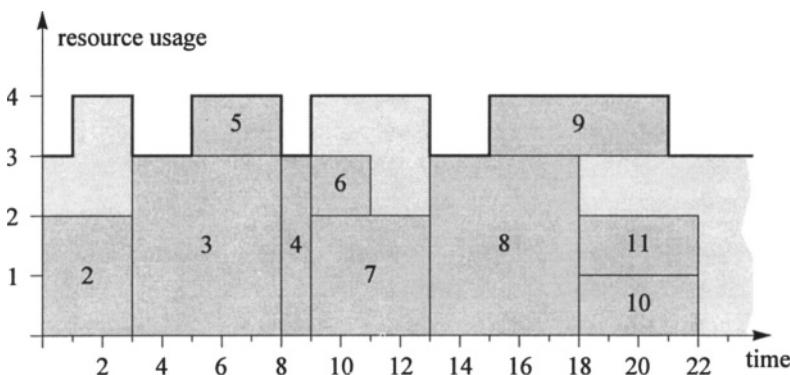


Figure 6.12. Optimal solution

After returning to the root node, the first alternative branch leads to the node 26 which is fathomed due to $LLB = 15 + \psi_8 = 15 + 9 = 24 \geq UB$. The same is true for the node 27. Subsequently, the optimal solution given in Figure 6.12 with $UB = 22$ is found in node 30. Due to $LLB = 22$ in the nodes 29 and 28, they can be fathomed. In the node 31, the core time rule applies. Back in the ancestor node 23, the initial local lower bound value of $LLB = 20$ can be increased to $LLB' = t'_{min} + \psi_6 = 13 + 11 = 24 \geq UB$ after removing job 6 from the stack. Note that the initial value of $t'_{min} = 11$ can be increased to 13, because job 6 can not be contained in any set E^* still to be branched. The node 32 is not developed due to $LLB = 12 + \psi_4 = 24$. The node 21 with $E^* = \{\}$ is fathomed by the active schedule rule because job 5 can always be (globally) left shifted to start at time point 5. For the node 20, the local lower bound value is increased to $LLB' = 4 + \psi_3 = 22$ after removing job 3 from the stack. The node 34 is eliminated due to $LLB = 6 + \psi_2 = 22$. Finally in the node 33, LLBM increases the local lower bound to 22 after taking job 3 from the stack. In the root node, $\{\}$ need not be branched because none of the resulting schedules can be active.

6.3 Scattered Branch and Bound

In this section, we introduce a new search strategy called *scattered branch and bound* which aims at overcoming some of the disadvantages inherent to the depth-first search strategies discussed in Section 6.1.2.

Consider a given branch and bound procedure for an optimization problem including a branching scheme and a set of bounding, reduction and dominance rules. Then for each problem instance to be solved there exists a *minimal tree* to be built in order to find an optimal solution and to prove its optimality having the least number of nodes among all enumeration trees possible (cf. Roucairol (1996)). In order to obtain this tree, the subproblems must be generated and considered in a particular sequence. Unfortunately, in general, such a sequence can not be predetermined, i.e., before having solved the problem instance to optimality. Hence, a lot of existing branch and bound approaches apply heuristic *search strategies* for determining a sequence which, hopefully, yields the optimal solution as soon as possible. In particular, this is true for branch and bound approaches which are also used as heuristic procedures by prematurely terminating them due to a restricted computation time. However, usually no effort is spent to gather information which can be used by dominance rules as effectively as possible.

A more general concept than applying a simple heuristic search strategy, called *target analysis*, is proposed by Glover (1986) and Glover and Greenberg (1989). It consists of classifying problem instances and determining a near-optimal search strategy for each class based on a learning approach. For each problem class, a set of training instances as well as different attributes (criteria) of subproblems for controlling the search process are defined. After solving the test instances to optimality, the sequences leading to the optimal solutions are evaluated and weights for the criteria are calculated such that subproblems are considered in these sequences. Having identified criteria and corresponding weights for each class, new instances are tackled by determining their problem class and controlling the search using the corresponding criteria-weights combination. Further concepts proposed in Glover (1986) have successfully been applied to the weighted tardiness problem by Fadlalla and Evans (1995). However, the concept of target analysis bears some difficulties. First of all, distinguishing different problem classes by using measures describing the properties of instances is difficult. This is, e.g., underlined by the extensive discussion of

measures for the resource-constrained project scheduling problem which aim at predicting the complexity of an instance in terms of computation time (cf. Section 7.2.1). Furthermore, a large amount of training with different instances from a variety of problem classes is required which constitutes a considerable effort. Finally, there remains the problem of choosing the right search strategy for problem instances which can not clearly be assigned to one of the problem classes.

Recognizing the importance of dominance rules in solving many combinatorial optimization problems (in particular, in the field of scheduling), Realff and Stephanopoulos (1998) have proposed an approach which, during the branching process, derives new dominance rules by an explanation-based learning algorithm from artificial intelligence. Unfortunately, as the authors state themselves, incorporating this concept into branch and bound algorithms usually will require a considerable effort. Furthermore, having identified all possible dominance rules for an optimization problem, there still remains the challenge of directing the search such that the minimal tree is obtained.

6.3.1 Principles of Scattered Branch and Bound

In this section, we present the principles of scattered branch and bound. The basic idea of this new search strategy consists of guiding the search process such that near-optimal solutions are yielded and most valuable information for dominance rules is gathered as soon as possible.

6.3.1.1 *A Critique of Traditional Branch and Bound*

The discussion above showed that, stipulating a given set of bounding, reduction and dominance rules, the effectiveness of branch and bound procedures depends on finding very good solutions soon and on obtaining favorable subproblems early. Therefore, many procedures using depth-first search strategies control the branching process heuristically. However, there always remains one disadvantage which is inherent to all procedures of this type. Despite whatever combination of branching scheme and search strategy is chosen, the search is performed in a very rigid manner which may result in the following difficulty. On early levels of the enumeration tree some disadvantageous branching decisions may have been made. Due to the depth-first search strategy, these branching decisions can not be revoked before completely having examined the cor-

responding subtree. This may require a considerable amount of effort which could be substantially reduced when an improved incumbent or dominance information provided in other parts of the enumeration tree were already on hand. Furthermore, when branch and bound procedures are prematurely terminated, e.g., because of limited computation time, they eventually have examined only such regions of the solution space containing rather poor solutions.

Example. Figure 6.13 contains the enumeration tree of Figure 6.1, p. 217 with the branches sorted according to the visiting order of DFSB (from the left to the right). Now assume that bounding and dominance rules are included and LB is the global lower bound. Dominance relationships between subproblems are indicated by dashed arcs (i,j) with i dominating j . After determining a solution with $UB = LB + 4$ in node 15, the search continues as given in Table 6.1, p. 218, except for the node 17 not being built due to fathoming its ancestor 8. A new incumbent with $UB = LB + 3$ is obtained in node 20. While examining the rest of the tree, the nodes 5, 6, and 4 are fathomed.

The *minimal tree* is obtained by considering branch j first. Continuing in an arbitrary order the nodes 2, 5, 6, and 4 are fathomed which makes building the nodes 7, 15, 16, 8, 17, 13, 12, 14, 11 obsolete. That is, the minimal tree contains 4 nodes less than the DFSB tree.

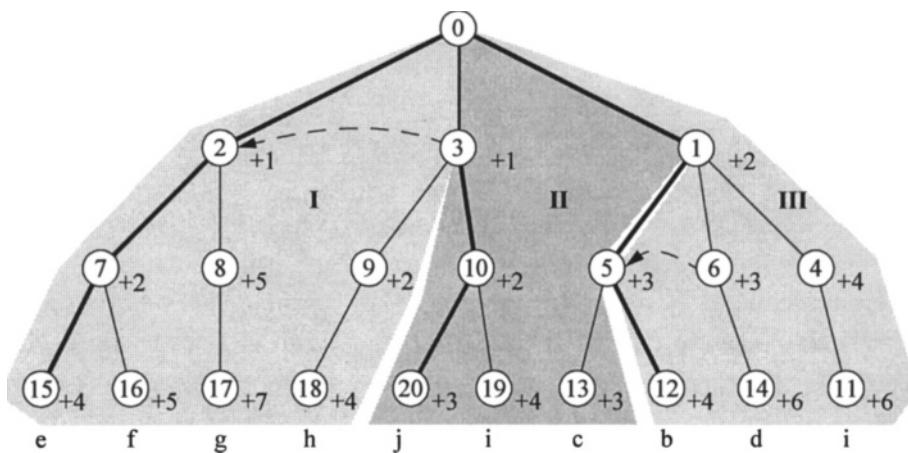


Figure 6.13. Enumeration tree according to DFSB

Due to the above reasons, heuristic meta-strategies such as simulated annealing, genetic algorithms and tabu search have emerged over the last two decades for solving complex combinatorial optimization problems (cf. Section 5.3). For

example, tabu search tries to overcome the rigidity of branch and bound approaches by performing moves, which transform one feasible solution into another and which allow for searching the solution space more flexibly than by branching (cf. Section 5.3.1). To prevent the search from cycling, moves with certain attributes are forbidden (set tabu), thus providing an adaptive form of memory. This basic concept is accompanied by intensification and diversification strategies which focus the search on promising regions or drive the search into new ones, respectively.

In the following, we propose a new search strategy for branch and bound procedures which exploits some of the ideas contained in tabu search. The basic idea consists of decomposing the search space into several regions which can be examined in an arbitrary order thus providing a possibility for scattering the search. It is designed for the use with depth-first search strategies because minimum lower bound strategies are becoming inapplicable for many complex combinatorial optimization problems due to computer storage restrictions.

6.3.1.2 Subdividing the Solution Space into Regions

In this section, we describe how the solution space of a combinatorial optimization problem can be decomposed into several disjoint regions such that these regions can be examined in a favorable sequence by a branch and bound procedure.

Assume a pool of feasible solutions, called *cut solutions*, to be given for a problem instance. Stipulating a branching scheme and a depth-first search strategy, for each of these solutions a corresponding branch of the enumeration tree can be constructed. Starting in the root node, this is done level by level always developing the node which represents the branching decisions leading to the leaf representing the cut solution. For the example in Figure 6.13, the pool could, e.g., contain the first solution obtained in leaf 15 as well as the solutions of the leafs 12 and 20. The corresponding branches are e, b, and j, respectively. Due to the given branching scheme and the search strategy, also the sequence in which these branches are arranged in the tree can be determined a priori. Supposing DFSB as search strategy, the sequence is e, j, b (cf. Figure 6.13).

Example. In Figure 6.13, those branches leading to one of the cut solutions are bold. As the figure shows, the branches naturally *cut* the enumeration tree into three different parts (subtrees). Each subtree is limited by a branch on the left as

well as on the right. The left-hand border always corresponds to the respective cut solution out of the pool. The right-hand border equals the last branch which, following the branching scheme and the search strategy, is considered just before obtaining the next cut solution or the right-hand border of the entire tree. Obviously, each *subtree* represents a certain *region* of the solution space. As a consequence, we use these terms synonymously. By having defined borders, the different subtrees can be examined in an arbitrary order. Note that neighboring subtrees may share parts of a branch as is the case with the subtrees II and III in Figure 6.13.

In the example, we could start with examining subtree II. After constructing the branch representing the cut solution j , the enumeration is continued as defined by the branching scheme and the search strategy until the left-hand border of subtree III is reached. This is true after having examined node 13 and returning to the ancestor node 5. Now, the search could be continued in subtree I before finally examining subtree III. By doing so, the minimal tree is obtained.

Static Generation. In general, cut solutions can be generated statically or dynamically. In case of a static generation, the pool of cut solutions is only determined once before starting the branch and bound procedure. Besides building the first branch of the enumeration tree, the most simple form of constructing cut solutions consists of repeatedly applying the branching scheme and randomly choosing a node to be developed on each level of the tree. Furthermore, problem-specific heuristics like, e.g., priority-rule based procedures can be used. However, due to their inherent structure, they bear the danger that cut solutions are yielded which are located all in the same region of the solution space. Therefore, it may be promising to obtain cut solutions by means of heuristic meta-strategies. For example, while performing a tabu search procedure, a set of *elite solutions* can be stored. Finally, one could think of more evolved approaches which assure that the cut solutions differ from each other by a desired "distance" such that, e.g., the solution space is divided into regions of approximately the same size.

Dynamic Generation. By dynamic generation, additional cut solutions may be inserted during performing the branch and bound procedure leading to a diversification of the search. This can be useful, when it becomes apparent that some regions have not yet been examined at all or that some subtrees are too large such that they should be further subdivided. Basically, all approaches described above can be used. However, if cut solutions are yielded which are lo-

cated in subtrees already examined, they have to be eliminated. Another approach is to perform a local search based on moves as tabu search does starting from elite branches, i.e., branches which led to a new incumbent or which were fathomed very late. One possibility to control this search consists of *path relinking*, a concept initially proposed by Glover (1989) for use within tabu search procedures (cf. also Glover and Laguna (1997, section 4.5)). The basic idea of this approach is to generate new cut solutions by exploring trajectories connecting elite solutions.

Besides defining the way of yielding cut solutions, their *number* is also an important parameter. Choosing the number too low may have the effect that some of the subtrees become too large. If favorable solutions or subproblems are located close to the right-hand border of these subtrees, they will be obtained rather late such that the advantage of partitioning the tree may get partially lost. By the way of contrast, if the number is chosen too large, the effort of frequently changing the subtrees may overcompensate the potential reduction of the enumeration tree. Therefore, no general recommendation on the number of cut solutions can be given without considering the structure of the optimization problem on hand.

Note that the concept of cut solutions also represents a technique which can be used for load balancing within branch and bound procedures designed for parallel computers. See Gendron and Crainic (1994) and Roucairol (1996) for current surveys on such procedures.

6.3.1.3 Swapping Regions

In general, it is not advantageous to examine a subtree and the corresponding region completely, i.e., without preemption. Instead, in order to diversify the search, it is generally more promising to swap between regions. In order to perform such swaps, a mechanism has to be provided which allows for interrupting the examination of a subtree and continuing later. This can be realized by storing the *branching decisions* which led to the node currently considered before a swap took place. When reentering the subtree, the branch leading to this node is easily reconstructed by following these branching decisions and the search process is continued as usual.

The two most important questions when swapping subtrees are which one to consider next and when to leave the current one. This can be controlled by se-

lection and *break-off* rules, respectively. For both types of rules, static as well as dynamic versions may be distinguished.

Selection Rules. *Static selection rules* define a sequence in which the subtrees are examined before starting the enumeration process. Each time a swap is performed the subtree considered next is already known, i.e., the current state of the search process is not taken into account. For example, the subtrees could be considered according to increasing objective function values of the cut solutions. *Dynamic selection rules* choose subtrees using information gathered when having visited them for the last time. Each time a swap occurs, e.g., the subtree with largest ratio of branched to fathomed nodes can be selected. For applying such rules, it can be useful to examine each subtree once at the beginning of the procedure. However, it has to be avoided when defining such rules that the search sets a focus only on a certain subset of all subtrees. One possibility to overcome this problem consists of storing attributes of branches, e.g., how often a variable is assigned a certain value. If for a certain number of branching operations, no new incumbent has been obtained, a *diversification* of the search can be realized by concentrating on subtrees which do not contain frequently occurring attributes. For this purpose, some frequency based memory may be applied as described for tabu search in Section 5.3.1.3, pp. 196.

Break-Off Rules. *Static break-off rules* interrupt the examination of a subtree according to criteria defined before starting the branch and bound procedure. For example, the maximum number of nodes which are built in each subtree before performing a swap can be limited. By the way of contrast, *dynamic break-off rules* decide whether to swap subtrees or not depending on the development of the search process. They may be further subdivided concerning the information they use. Simple rules may restrict to information on the subtree currently evaluated. For example, they can refer to the number of nodes without improvement or compare the number of nodes branched to the number of nodes fathomed. As soon as the ratio falls below a predefined level, a swap is performed. Otherwise, the search is continued. More evolved rules may try to combine dynamic selection and the break-off mechanism. For example, as soon as the ratio of branched to fathomed nodes of the current subtree falls below the largest ratio obtained for any of the other subtrees already visited, a swap to the respective subtree is initiated. In order to prevent the search from becoming nervous, i.e., changing subtrees too often, a minimum number of nodes which must at least be evaluated in each subtree before swapping should be defined.

6.3.2 SCATTER: Scattered Branch and Bound for GRCPSP

After describing the principles of scattered branch and bound in the previous section, we now show, how this search strategy can be incorporated within the procedure PROGRESS presented in Section 6.2.

6.3.2.1 Outline

Before discussing the generation of cut solutions as well as the selection and break-off rules applied within SCATTER in detail, we describe the general framework used for intensifying and diversifying the search process (cf. Figure 6.14). Furthermore, we sketch how the data used for scattering the search is stored. To ease the description, we refer to the enumeration tree obtained by the parallel branching scheme and LLBM as the *reference tree* (cf. Figure 6.10, p. 237).

At the beginning, a set of cut solutions ($i = 1, \dots, v$) is generated statically for the problem instance considered. As described in Section 6.3.1.2, the branch leading to each cut solution represents the left-hand border of a subtree. For each subtree $i = 1, \dots, v$ a predefined number γ of subproblems is developed in order to gather some initial control information required for applying dynamic selection rules. The sequence of this initial examination is controlled by a static selection rule. If, after this preprocessing, all subtrees have already been

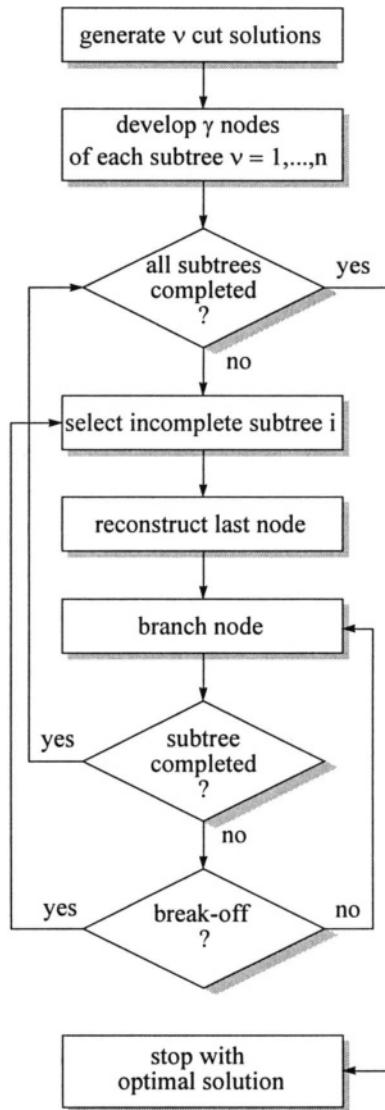


Figure 6.14. Framework of SCATTER

evaluated completely, the procedure stops and the incumbent represents an optimal solution of the problem instance considered.

Otherwise, an(other) incomplete subtree i is selected by a dynamic selection rule. The node in which the prior examination stopped (the last time) is reconstructed and branching continues in the usual manner. Note that if during the reconstruction a node is obtained which can be fathomed, e.g., due to improving the incumbent in the meantime, the reconstruction is stopped and the first alternative branch of its ancestor is developed. The branching stops when either a break-off rule decides to diversify the search or the subtree is completed. If still some incomplete subtrees exist, one of them chosen.

To realize SCATTER, a candidate list with an entry for each subtree i is defined containing the following information. The branching decisions leading to the corresponding cut solution i as well as the ones yielding the node examined when breaking-off for the last time are stored (left-hand and right-hand border). Initially, the left-hand and the right-hand border are equal. A subtree i is completely examined when the branching decisions match the ones corresponding to the left-hand border of the subtree i' next to the right in the reference tree. Finally, some control information on the development of the search process in subtree i of the tree is stored (cf. Section 6.3.2.3) which is required by dynamic selection rules and is updated each time a subtree is visited.

6.3.2.2 Decomposing the Solution Space

SCATTER generates cut solutions statically, i.e., before starting the branch and bound process. As the first cut solution we take the most-left branch of the reference tree. Further $\mu \geq v - 1$ cut solutions are computed by applying the branching scheme with randomly choosing one of the subproblems to be developed in each node. Though this approach is quite simple, it has the advantage that the branching decisions, i.e. the jobs scheduled on each level, which lead to a cut solution need not be reconstructed but can be stored during the generation.

Generating more cut solutions than being used allows for sorting these solutions according to increasing objective function values and taking the best $v - 1$ ones. Furthermore, duplicates have to be removed. Whenever the project completion time of a cut solution is equal to the global lower bound, this solution is optimal and the procedure stops.

Example. Consider our example given by the start-to-start project network in Figure 6.2, p. 221, and the resource availability profile in Figure 6.3, p. 223. With $\mu = 2$ and $v = 2$, the generated cut solutions could be the ones obtained by the branches a, b, and c given in Figure 6.15 with branch a being the most-left branch of the reference tree in Figure 6.10, p. 237. The two other ones are randomly generated as described above. The corresponding project completion times are 24, 24 and 27 respectively. Hence, the solution yielded by branch c would be ignored.

6.3.2.3 Swapping Regions

When swapping regions, both, some selection rule and some break-off rule, have to be defined. Within SCATTER, the following dynamic rules turned out to perform best.

Dynamic Selection Rule. It aims at choosing such subtrees first for which the probability that their examination will lead to a new incumbent is as large as possible. This probability is assumed to depend on the sum of the *total slack times* of the non-dummy jobs $j = 2, \dots, n-1$. For a certain project completion time, e.g., the planning horizon \bar{T} , the total slack time of a job is the difference between the latest and the earliest (or scheduled) starting time. The total slack time of a job which is obtained by a forward and a backward pass considering a partial schedule PS is defined as follows (cf. Section 2.2.2, pp. 46):

$$TSL_j(PS) = LS_j(\bar{T}) - SS_j(PS) \quad \text{for } j \in J(PS) \quad (6.1)$$

$$TSL_j(PS) = LS_j(\bar{T}) - ES_j(PS) \quad \text{for } j \in J - J(PS) \quad (6.2)$$

<table border="1" style="margin: auto;"> <tr><td>0</td><td>0</td></tr> <tr><td>{3}</td><td></td></tr> <tr><td>1</td><td>5</td></tr> <tr><td>{2}</td><td></td></tr> <tr><td>2</td><td>8</td></tr> <tr><td>{4}</td><td></td></tr> <tr><td>3</td><td>9</td></tr> <tr><td>{6}</td><td></td></tr> <tr><td>4</td><td>10</td></tr> <tr><td>{5}</td><td></td></tr> <tr><td>5</td><td>11</td></tr> <tr><td>{8}</td><td></td></tr> <tr><td>6</td><td>15</td></tr> <tr><td>{9}</td><td></td></tr> <tr><td>7</td><td>16</td></tr> <tr><td>{7, 10}</td><td></td></tr> <tr><td>8</td><td>20</td></tr> </table>	0	0	{3}		1	5	{2}		2	8	{4}		3	9	{6}		4	10	{5}		5	11	{8}		6	15	{9}		7	16	{7, 10}		8	20	<table border="1" style="margin: auto;"> <tr><td>0</td><td>0</td></tr> <tr><td>{2}</td><td></td></tr> <tr><td>1</td><td>3</td></tr> <tr><td>{3}</td><td></td></tr> <tr><td>2</td><td>5</td></tr> <tr><td>{5}</td><td></td></tr> <tr><td>3</td><td>8</td></tr> <tr><td>{4}</td><td></td></tr> <tr><td>4</td><td>9</td></tr> <tr><td>{6, 7}</td><td></td></tr> <tr><td>5</td><td>11</td></tr> <tr><td>{10}</td><td></td></tr> <tr><td>6</td><td>13</td></tr> <tr><td>{9}</td><td></td></tr> <tr><td>7</td><td>15</td></tr> <tr><td>{8}</td><td></td></tr> <tr><td>8</td><td>20</td></tr> </table>	0	0	{2}		1	3	{3}		2	5	{5}		3	8	{4}		4	9	{6, 7}		5	11	{10}		6	13	{9}		7	15	{8}		8	20	<table border="1" style="margin: auto;"> <tr><td>0</td><td>0</td></tr> <tr><td>{4}</td><td></td></tr> <tr><td>1</td><td>1</td></tr> <tr><td>{3}</td><td></td></tr> <tr><td>2</td><td>6</td></tr> <tr><td>{2, 6}</td><td></td></tr> <tr><td>3</td><td>8</td></tr> <tr><td>{10}</td><td></td></tr> <tr><td>4</td><td>10</td></tr> <tr><td>{9}</td><td></td></tr> <tr><td>5</td><td>11</td></tr> <tr><td>{5}</td><td></td></tr> <tr><td>6</td><td>14</td></tr> <tr><td>{7}</td><td></td></tr> <tr><td>7</td><td>18</td></tr> <tr><td>{8}</td><td></td></tr> <tr><td>8</td><td>23</td></tr> </table>	0	0	{4}		1	1	{3}		2	6	{2, 6}		3	8	{10}		4	10	{9}		5	11	{5}		6	14	{7}		7	18	{8}		8	23
0	0																																																																																																							
{3}																																																																																																								
1	5																																																																																																							
{2}																																																																																																								
2	8																																																																																																							
{4}																																																																																																								
3	9																																																																																																							
{6}																																																																																																								
4	10																																																																																																							
{5}																																																																																																								
5	11																																																																																																							
{8}																																																																																																								
6	15																																																																																																							
{9}																																																																																																								
7	16																																																																																																							
{7, 10}																																																																																																								
8	20																																																																																																							
0	0																																																																																																							
{2}																																																																																																								
1	3																																																																																																							
{3}																																																																																																								
2	5																																																																																																							
{5}																																																																																																								
3	8																																																																																																							
{4}																																																																																																								
4	9																																																																																																							
{6, 7}																																																																																																								
5	11																																																																																																							
{10}																																																																																																								
6	13																																																																																																							
{9}																																																																																																								
7	15																																																																																																							
{8}																																																																																																								
8	20																																																																																																							
0	0																																																																																																							
{4}																																																																																																								
1	1																																																																																																							
{3}																																																																																																								
2	6																																																																																																							
{2, 6}																																																																																																								
3	8																																																																																																							
{10}																																																																																																								
4	10																																																																																																							
{9}																																																																																																								
5	11																																																																																																							
{5}																																																																																																								
6	14																																																																																																							
{7}																																																																																																								
7	18																																																																																																							
{8}																																																																																																								
8	23																																																																																																							
a	b	c																																																																																																						

Figure 6.15. Cut solutions

The selection rule may proceed as follows. Each time, a break-off occurs in a subtree i , the sum of all total slack times $SSL_i = \sum_{j=2}^{n-1} TSL_j(PS)$ of the currently considered partial schedule PS is computed and stored in the corresponding entry of the candidate list. When selecting the next subtree to be evaluated, the one with the largest value of SSL_i is chosen.

In order to get information not only for a small part of a subtree i , the total slack time SSL_i as a measure for judging i is only updated at lower levels of a subtree. As a convention, the updating is restricted to the lower half of the subtree taking the maximal number of levels necessary for constructing the cut solutions as height of the tree.

The value of SSL_i can already be computed for the cut solutions and therefore also be used for determining the sequence in which the subtrees are initially examined for gathering control information. Furthermore, it is important to state that the rule also reflects the degree of utilizing the resources before the decision point t of a partial schedule PS . The smaller the value of SSL_i is, the more jobs have been scheduled early thereby utilizing the resources effectively.

Dynamic Break-Off Rule. It tries to consider the current state of the search process in order to decide whether to intensify the search, i.e., to continue branching in the current subtree, or to diversify it by swapping to another subtree. Using such a rule requires to provide appropriate measures which are able to describe the state of the search process. Within SCATTER the given measures below are recorded when examining a subtree $i = 1, \dots, v$. Each time a subtree is reentered, the values of the measures are set to 0. The same is true, when the decision is made to continue the search in the current subtree instead of diversifying it.

- nb_i : number of nodes *branched* in subtree $i = 1, \dots, v$
- nf_i : number of nodes *fathomed* in subtree $i = 1, \dots, v$

Besides these measures referring to the last visit of a subtree only, the number of times (iterations) it_i each subtree $i = 1, \dots, v$ has been entered so far is stored. Note that $it_i = 1$ when subtree i is considered the first time and that it_i is only increased when the subtree is reentered.

With these measures, the break-off rule can be described as follows. In order to prevent the search from becoming nervous, i.e., changing subtrees too often, at least a minimum number δ of nodes must be evaluated each time a subtree is

visited. This is achieved by deciding about breaking-off only if the condition $nb_i + nf_i > \delta \cdot it_i$ is fulfilled. The rationale behind multiplying δ with it_i consists of intensifying the search depending on the development of the search process. If it_i is high, this is due to selecting the subtree i rather often by the selection rule and, hence, its examination seems to be promising.

In case the above condition is fulfilled, the decision whether to break-off or not still has to be made. The current subtree is left, when the *branching ratio* nb_i / nf_i falls below 1. That is, the search is *diversified* when more nodes are fathomed than branched. Vice versa, as long as more subproblems are branched than fathomed, it is assumed that continuing with branching may succeed in finding a new incumbent. Hence, the search is *intensified* by staying in the current subtree.

Remark 6.2. Note that when swapping regions the schedule storing rule must be applied carefully. In particular, it must not be used when reconstructing a branch due to reentering a subtree.

6.3.2.4 Example

Figure 6.16 contains the enumeration tree obtained by SCATTER when solving our example instance defined by the start-to-start project network in Figure 6.2, p. 221, and the resource availability profile in Figure 6.3, p. 223. For a planning horizon of $\bar{T} = 30$, we assume a constant availability $a_{1t} = 3$ in the periods $t = 23, \dots, 30$. With $v = 2$, we choose the cut solutions 1 and 2 corresponding to the ones yielded by the branches a and b of Figure 6.15. The shaded nodes a2 to a8 represent the ones of the cut solution 1 (branch a) which are not reconstructed by SCATTER due to successful fathoming. At most 8 levels are considered when generating the cut solutions. Hence, when computing SSL_i for $i = 1$, the partial schedule PS being obtained in node 4 with $t = 10$ and $J(PS) = \{3, 2, 4, 6\}$ is considered. For $i = 2$, the schedule PS with $t = 9$ and $J(PS) = \{2, 3, 5, 4\}$ is used. The values yielded while computing SSL_1 and SSL_2 are given in Table 6.5.

Due to $SSL_1 = 104$ and $SSL_2 = 114$, we start with examining subtree 2 for gathering control information. After finding a solution with $UB = 24$ in node 8 and fathoming node 9 by computing $LLB = 24 \geq UB$, the optimal solution with $UB = 22$ is found in node 12. Subsequently, the nodes 11, 10, 13, 14 are not further developed according to the corresponding nodes 29, 28, 31, 32 of the

Table 6.5. Calculation of slack times ($\bar{T}=30$)

j		2	3	4	5	6	7	8	9	10	11	SSL _i
LS _j (\bar{T})		14	12	18	19	19	22	21	24	26	26	
i = 1	ES _j (PS), SS _j (PS)	5	0	8	10	9	13	11	13	11	17	
	TSL _j (PS)	9	12	10	9	10	9	10	11	15	9	104
i = 2	ES _j (PS), SS _j (PS)	0	3	8	5	9	9	12	13	11	17	
	TSL _j (PS)	14	9	10	14	10	13	9	11	15	9	114

reference tree in Figure 6.10, p. 237. By evaluating the nodes 16 and 17 with the latter being fathomed by calculating $LLB = 6 + \psi_2 = 22$, subtree 2 is already examined completely during the first visit intended for recording control information, which is due to the small size of the example. While reconstructing branch a, job 3 is scheduled to start at time point 0 yielding node 15. In this node the core time rule applies for the jobs 7 and 8. For $UB = 22$, the latest starting times are $LS_7(UB - 1) = 13$ and $LS_8(UB - 1) = 12$. For job 7, we yield $EF_7(PS) = 17$ due to job 2 which can not start before the current decision point $t = 5$. Job 8 can not finish before $EF_8(PS) = 14$ because of the minimum time lag $\lambda_{38} = 9$. Hence, the two jobs have to be processed concurrently at least between the time points 13 and 14 (period 14) which is impossible due to the resource constraints.

6.4 Existing Procedures

In this section, we present a survey on other existing branch and bound procedures which have been designed for solving RCPSP and GRCPSP. In particular for RCPSP, a large number of such approaches have been proposed. Therefore, describing all of them in detail is not possible. However, a careful analysis reveals that the procedures mainly differ concerning the incorporated branching schemes. Besides this major characteristic, further differences are due to the search strategy applied as well as to additional components included such as the lower bound arguments and the dominance rules. Since the latter topics have already been discussed extensively in the previous sections, the survey is organized along the branching schemes restricting to those ones which have turned out to be most promising. For each branching scheme, the search strategy as well as the most important bounding and dominance rules included in the best performing procedure are presented in more detail. Further branch and bound

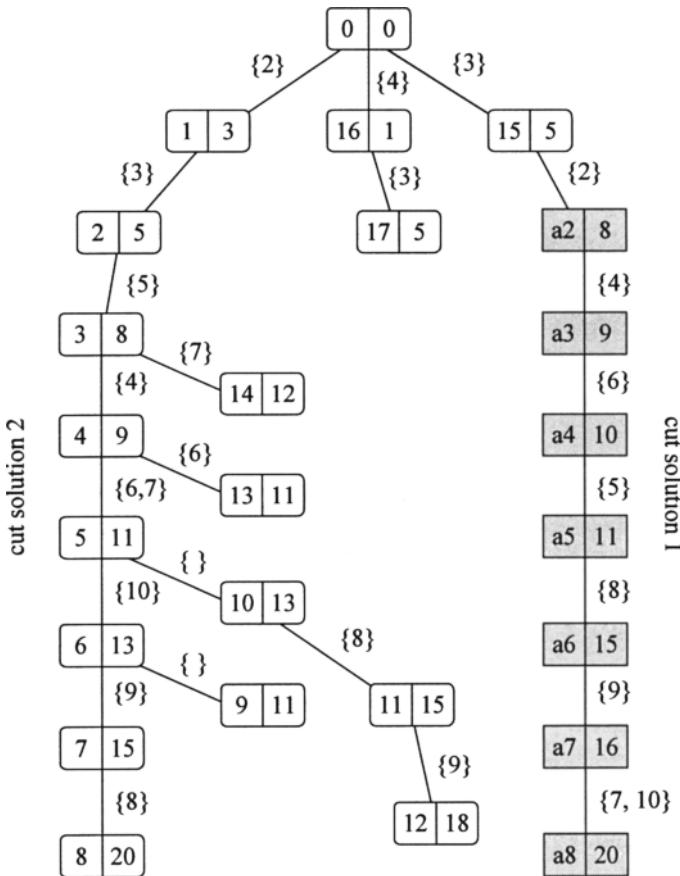


Figure 6.16. Enumeration tree of SCATTER

algorithms for RCPSP based on alternative branching schemes are, e.g., suggested by Müller-Meerbach (1967 a, 1967 b), Balas (1970), Davis and Heidorn (1971), Talbot and Patterson (1978), Radermacher (1985/86), and Carlier and Latapie (1991). Simpson and Patterson (1996) extend the approach presented by Talbot and Patterson (1978) such that it can be applied on parallel computers. Furthermore, they consider fluctuations in the resource availability.

6.4.1 Parallel Branching Scheme

The parallel branching scheme has already extensively been discussed for GRCPSP in Section 6.2.1 such that a description is omitted. The only modification required for an application to RCPSP concerns the simplified computation

of the next decision point $t' > t$ which corresponds to the smallest scheduled finishing time of all jobs $j \in J(PS)$ not being active in period $t+1$. Within RCPSP procedures, the parallel branching scheme has been applied by Johnson (1967), Hastings (1972), and Stinson et al. (1978). Furthermore, Mingozi et al. (1998) have employed a branching scheme which differs only slightly from the parallel one. Indeed, analyzing both scheduling schemes carefully reveals that feasible partial schedules are obtained in the same order. Therefore, we renounce a separate description.

Among the first three procedures the one of *Stinson et al. (1978)* is the most efficient (cf. Patterson (1984)). The search strategy used in this procedure represents some variant of the MLB strategy described in Section 6.1.2, pp. 216. However, some compound criterion is used for selecting the subproblem to be developed next from the global candidate list. Besides the local lower bound value of a subproblem, this criterion also includes data such as the number of jobs scheduled, the current decision point t , and the number of jobs becoming available (cf. Stinson et al.(1978)). For fathoming subproblems, the procedure employs the critical path bound LBC1, the capacity bound LBC2 and the critical sequence bound LBC3 (cf. Section 4.1.1.1). Furthermore, the semi-active schedule rule as well as a special version of a scheduling storing rule are included (cf. Sections 6.2.4.2 and 6.2.4.4, respectively).

Also the procedure of *Mingozi et al. (1998)* applies a variant of the MLB strategy. However, the subproblems in the global candidate list are sorted according to non-decreasing numbers of jobs contained in the corresponding partial schedules instead of the respective local lower bound values. For this reason, the enumeration tree is constructed almost like when using the DFSB strategy, because subproblems located on higher levels are considered first. For fathoming subproblems, the node packing bound LBN1 is computed in each node of the enumeration tree (cf. Section 4.1.1.3, pp. 120). Furthermore, a left shift rule ensures that only semi-active schedules are evaluated. Finally, the cutset rule of Demeulemeester and Herroelen (1992) which represents an RCPSP variant of the simple schedule storing rule is incorporated (cf. Section 6.4.3).

6.4.2 Serial Branching Scheme

This branching scheme which is used in the procedure GSA (general sequencing algorithm) of Sprecher (1997) has been derived from an algorithm of Patterson

et al. (1989) designed for multi-mode RCPSP. It is closely related to the serial scheduling scheme for priority rule based heuristics (cf. Section 5.2.1.1, pp. 169) and has originally been described as follows. According to the parallel branching scheme discussed above, each node of the enumeration tree represents a feasible partial schedule PS. However, descending nodes are obtained by assigning a starting time only to a single available job instead to a subset of eligible ones. For this purpose, the set of available jobs A(PS) is initially determined in each node (cf. Definition 5.2, p. 164). Subsequently, for each of these jobs $j \in A(PS)$, a descending node is constructed as follows (with the jobs being considered according to increasing job numbers). After computing the schedule dependent earliest starting time $ES_j(PS)$, job j is scheduled to start at the smallest time point $SS_j \geq ES_j(PS)$ for which its execution does not violate the resource constraints in each of the periods $SS_j + 1, \dots, SS_j + d_j$. That is, each branch of the tree corresponds to a precedence feasible permutation of the jobs. Note that the jobs can initially be renumbered according to some priority rule such that the topological sorting of the jobs is preserved and the descending nodes are obtained within an advantageous order.

One disadvantage of this basic version consists of the circumstance that identical schedules may be considered in different branches of the tree (cf. Remark 5.11, p. 200). To partially overcome this disadvantage, Sprecher (1997) has proposed the following modification. In each descending node, no job $j \in A(PS)$ is allowed to start earlier than the job h scheduled in the ancestor node, i.e., the condition $SS_j \geq SS_h$ is additionally introduced. Furthermore, each node is immediately fathomed, if a job j is assigned the same starting time as a job $h > j$. In this case, the resulting partial schedule has already been examined in one of the neighbor branches where the two jobs have been considered in the reverse order.

The procedure of *Sprecher (1997)* uses DFSL as search strategy. In order to yield promising subproblems first, the jobs are renumbered according to non-decreasing earliest finishing times obtained by a simple forward pass. For fathoming purposes, a number of dominance rules are included among which the following ones are the most important. First of all, left shift rules ensure that only active schedules are considered by verifying whether the job scheduled last can be locally or globally left shifted (cf. Section 6.2.4.2). These rules are complemented by the contraction rule which prevents from constructing some of the partial schedules which would be fathomed by the left shift rules on one

of the following levels of the enumeration tree. Furthermore, a variant of the simple schedule storing rule described in Section 6.2.4.4 is used which considers the current partial schedule PS as well as previously evaluated ones \bar{PS} . Assume that the next descending node with the schedule PS' is obtained from PS by scheduling job j at $SS_j(PS')$. Then, the schedule PS' can be discarded when the following conditions are fulfilled:

- In both partial schedules PS and \bar{PS} , the same set of jobs is scheduled, i.e., $J(\bar{PS}) = J(PS)$.
- The scheduled starting time of job j in PS' is not smaller than the maximum finishing time of any job in \bar{PS} , i.e., $SS_j(PS') \geq \max\{SF_h(\bar{PS}) \mid h \in J(\bar{PS})\}$. In this case, none of the scheduled jobs $h \in J(\bar{PS})$ is active in one of the periods $SS_j(PS') + 1, \dots$

This variant of the simple schedule storing rule has the advantage that the information required can be stored efficiently. For each non-dominated partial schedule which has been evaluated so far, only the jobs considered as well as the maximum scheduled finishing time of any of these jobs has to be recorded. However, this rule can not be transferred to GRCPSP (cf. p. 233). Finally, a simple permutation rule and a non-optimality rule are included. In addition to these dominance rules, the critical path bound LBC1 and a variant of the node packing bound LBN1 are employed (cf. Sections 4.1.1.1, pp. 114, and 4.1.1.3, pp. 120). In this variant, rotation of the priority list is omitted. Instead, three different lists are constructed by applying different priority rules.

6.4.3 Delaying Alternatives

This branching scheme represents the one most frequently used within RCPSP procedures and is employed in some variations which differ only slightly. In the following, we describe the version proposed by Demeulemeester and Herroelen (1992) because the corresponding procedure is considered to be one of the most efficient on hand for solving RCPSP (cf. Demeulemeester and Herroelen (1997 b)). Furthermore, this version is also incorporated in the GRCPSP procedure of Demeulemeester and Herroelen (1997 a). Further procedures are, e.g., described by Christofides et al. (1987), Bell and Park (1990), and Nazareth et al. (1999). Note that the procedure of Christofides et al. (1987) may terminate without having found an optimal solution. A corrected version is proposed in Demeulemeester et al. (1994).

According to the parallel branching scheme, each node of the enumeration tree corresponds to a feasible partial schedule PS with an associated decision point t . Before branching a node, the set of eligible jobs $E(PS,t)$ which can be started at time point t without violating precedence or resource constraints is determined (cf. Section 6.2.1, pp. 222). Then, *all* jobs $j \in E(PS,t)$ are scheduled at time point t by way of trial in addition to the active ones. If no resource conflict occurs in the following periods $t+1, \dots$ only one descending node exists. Otherwise, branching is performed by determining all minimal *delaying alternatives*. A delaying alternative is a subset $D(PS,t)$ of those jobs active and eligible at t the delaying of which resolves the resource conflict at t . It is called minimal if no other delaying alternatives $D'(PS,t)$ with $D'(PS,t) \subset D(PS,t)$ exists. For each minimal delaying alternative $D(PS,t)$, a descending node is obtained by unscheduling all jobs $j \in D(PS,t)$ again. The next decision point $t' > t$ is computed as the smallest scheduled finishing time of all jobs $j \in J(PS)$ active in period $t+1$. Note that though this branching scheme is quite similar to the parallel one, both scheduling schemes are obviously not identical (cf. Sprecher (1998) for a simple example).

Within the RCPSP procedure of *Demeulemeester and Herroelen (1992)*, the DFSB search strategy is employed (cf. Section 6.1.2). For this purpose, the critical sequence bound LBC3 is applied for each descending node (cf. Section 4.1.1.1). Besides this bounding rule, the procedure contains the following dominance rules. First of all, a left shift rule is used in order to restrict the enumeration to semi-active schedules. However, this rule is not sufficient, i.e., schedules which are not semi-active may be constructed nevertheless (cf. Sprecher and Drexl (1996)). Two other rules identify partial schedules for which certain eligible jobs can immediately be scheduled at the current decision point t without considering any alternative descending nodes. For example, this is true, if no job is active at the current decision point t , and there exists a job among the eligible ones which is incompatible to all other unscheduled jobs. Finally, a schedule storing rule called *cutset rule* is included. This rule compares the current partial schedule PS to previously considered ones \bar{PS} . The schedule PS is not examined, if the following conditions are true:

- The set of jobs scheduled in PS equals those of \bar{PS} , i.e., $J(PS) = J(\bar{PS})$.
- The decision point t associated with PS is not smaller than the one \bar{t} of \bar{PS} , i.e., $t \geq \bar{t}$.

- Each job $j \in J(\bar{PS})$ which is active at the decision point \bar{t} ($SF_j(\bar{PS}) > \bar{t}$) does not finish later than the maximum of the decision point t of PS and its corresponding finishing time, i.e., $SF_j(\bar{PS}) \leq \max\{t, SF_j(PS)\}$.

Recently, *Demeulemeester and Herroelen (1997 b)* have presented an improved version of their algorithm. The critical sequence lower bound LBC3 is replaced by the node packing bound LBN1 (cf. Section 4.1.1.3). Furthermore, the code has been adapted such that it fully exploits the possibilities provided by modern 32-bit operating systems. For example, this allows storing the data required by the cutset rule much more efficiently (cf. Section 7.1, pp. 262).

Within the procedure of *Demeulemeester and Herroelen (1997 a)* developed for GRCPSP, the following modifications have been made. The next decision point is determined as described for the parallel scheduling scheme in Section 6.2.1. Only the critical path based lower bound LBC1' is used (cf. Section 4.3.1.1, pp. 152). Furthermore, all dominance rules but the left shift one and the cutset one are excluded. The latter one is adapted by complementing the following conditions which have to be fulfilled in addition to the above ones for verifying a dominance relationship:

- The earliest starting times of all unscheduled jobs $j \in J - J(PS)$ fulfill $ES_j(\bar{PS}) \leq ES_j(PS)$.
- The per period availability does not change for any resource type in the periods $q \in [\bar{t} + 1, t]$.

6.4.4 Schedule Schemes

This branching scheme has been introduced by Brucker et al. (1998) for RCPSP (cf. also Brucker and Knust (1999)). In contrast to the schemes presented so far, each node of the enumeration tree corresponds to a set of feasible schedules instead of a single partial one. The respective sets of schedules are described by *schedule schemes* which can be motivated as follows.

Assume a feasible schedule CS. Then, for each pair of jobs (h, j) one of the following relations holds in CS. If the execution of both jobs (partially) overlaps, a *parallelity relation* $h \parallel j$ is present. Otherwise, either job h has to be finished before job j or vice versa and a *conjunction* $h \rightarrow j$ or $j \rightarrow h$ is induced. In order to describe a set of schedules, some of these relationships may be relaxed. Conjunctions may be substituted by *disjunctions* $h \leftrightarrow j$ which indicate that the jobs

h and j must not be processed in parallel. Furthermore, parallelity relations and disjunctions may be replaced by *flexibility relations* $h \sim j$. For a pair (h,j) , such a relation implies that no decision has been made yet whether a disjunction or a parallelity relation is established. That is, a *schedule scheme* defines one the above four relations for each pair of jobs (h,j) . With CR, DR, PR, and FR representing the sets of conjunctions, disjunctions as well as parallelity and flexibility relations, respectively, it can be denoted by (CR, DR, PR, FR) .

In the root node of the enumeration tree, the initial schedule scheme can be obtained as follows. A conjunction is established for each pair (h,j) where a job h has to be processed before a job j due to a (transitive) precedence relationship, whereas a disjunction is set up for each pair (h,j) which is incompatible because of the resource constraints. For all remaining pairs, a flexibility relation is assumed by default, i.e., the execution of the jobs is not subject to any obvious restrictions. In each subproblem, a single pair of jobs (h,j) for which a flexibility relation exists is selected. For this pair, two descending nodes are constructed by replacing the flexibility relation by a disjunction and a parallelity relation, respectively, such that a binary enumeration tree is obtained.

As soon as the set FR is empty, a leaf of the enumeration is yielded. Though the respective schedule scheme may still describe a set of schedules, one with the smallest project completion time among them can be computed by a special purpose procedure in $O(m \cdot n^3)$ time. However, also no feasible schedule may exist at all which is detected by the same procedure.

The search strategy employed in the procedure of *Brucker et al. (1998)* is DFSB. In each node of the enumeration tree, the set covering based lower bound for RCPSP is computed using the column generation approach of Baar et al. (1998) (cf. Section 4.1.2.3, pp. 134). In order to select the next pair of jobs with a flexibility relation for branching, the resulting lower bound values of the subproblems are estimated using information which has been provided by calculating the lower bound value of the current subproblem. In addition to fathoming by bounding, the procedure contains a large number of reduction rules the application of which is referred to as immediate selection. They all have in common, that they aim at fixing additional conjunctions, disjunctions and parallelity relations by applying some logical tests. Since these rules are rather complex, we renounce giving a detailed description.

7 Computational Experiments

This chapter presents comprehensive computational experiments examining the computational efficiency of lower bound arguments and solution procedures for resource-constrained project scheduling.

The purpose behind executing such experiments is twofold. First of all, most of the procedures described in this book can be configured in different ways with the chosen configuration having considerable influence on their computational efficiency. For example, the destructive improvement technique may be combined with different bound arguments. Within PROGRESS and SCATTER only a subset of all dominance rules and logical tests may be applied. Therefore, it has to be determined which configuration of each procedure yields the best results.

Secondly, a large number of procedures for resource-constrained project scheduling have been presented in the literature so far, including lower bound arguments, heuristic approaches as well as branch and bound methods (cf. Chapters 4 to 6). In order to validate the efficiency of the procedures introduced in this book, comprehensive evaluations are performed. For this purpose, each type of procedure is compared to the most efficient existing ones.

Except for the lower bound arguments, computational results are presented for both, RCPSP and GRCPSP. For RCPSP, a number of challenging benchmark data sets have been proposed in the literature. By the way of contrast, this is not the case for GRCPSP. Therefore, new data sets for this problem type are systematically generated based on the RCPSP ones.

The chapter is organized as follows. In Section 7.1, the computational environment which is used for performing the studies is described. Section 7.2 is devoted to the description of complexity measures and benchmark data sets. In the remaining Sections 7.3, 7.4, and 7.5, the efficiency of lower bound arguments, heuristic approaches, and exact procedures is examined, respectively.

7.1 Hardware and Software Environment

All computational experiments concerning procedures described in this book have been performed on a single IBM-compatible personal computer which has been equipped as follows:

- Intel Pentium central processing unit with 166 MHz clock pulse
- 64 MB random access memory with 60 nano seconds access time
512 KB pipelined burst cache
- Windows 95 32-bit operating system

The procedures have been coded using ANSI standard C and compiled by means of Borland C++ 5.0 (cf. Borland (1996)). In Section 7.4.1.4, computational results are reported which have been obtained by proprietary heuristics of project management software packages. The packages considered are Computer Associates' SuperProject 4.0a and Microsoft's Project 4.0. Note that all given product names are registered trademarks.

All input data, i.e., precedence relationships, durations, resource usages and availabilities are stored in standard arrays. The same is true for algorithmic and solution data. In order to keep the storage requirements for resource profiles as small as possible, the availability (usage) is not stored for every period and resource type explicitly. Instead, only those periods as well as the corresponding availabilities (usages) are kept in memory where fluctuations occur. That is, the memory required for holding a resource profile depends on the number of fluctuations instead of the length of the planning horizon. Though representing an increased update effort for instances with a small project completion time, the major advantage of this concept is that the computational effort for updating data is not changed by scaling the duration of jobs and lengths of resource profiles.

In general, no effort is spent to speed up the codes by explicitly exploiting the 32-bit architecture of modern operating systems such as Windows 95, Windows NT, and Linux as proposed in Demeulemeester and Herroelen (1997 b) and adapted in Sprecher (1997) for exactly solving RCPSP. Within the according procedures, the data of four resource types, e.g., the availabilities in a certain period or the resource usages of a job, are combined into one single 32-bit integer variable which allows for a much more efficient checking of the re-

source constraints. However, this has the disadvantage that for storing the data of each resource type only 8 bit are available and, hence, the maximum value which can be considered is 255. Therefore, within our procedures only standard integer variables are used for storing data allowing for maximum values of $2^{32}-1$. Furthermore, accelerating a procedure by "code polishing" as advocated by Demeulemeester and Herroelen (1997 b) does not change its general strengths and weaknesses. In particular, this is true because resource-constrained project scheduling problems are NP-hard (partly in the strong sense). For further 32-bit specific features see Demeulemeester and Herroelen (1997 b) and Sprecher (1997).

7.2 Complexity Measures and Data Sets

A number of benchmark data sets have been proposed for RCPSP in the literature so far. In most of the recent publications, the computational experiments refer to data sets which have been generated using the problem generator PROGEN developed for RCPSP by Kolisch et al. (1995). The generation of instances using PROGEN is controlled by specifying values for different complexity measures which aim at predicting the complexity of instances in terms of computation time. Since these complexity measures also help to examine the performance of RCPSP procedures more systematically, they are briefly discussed in Section 7.2.1, before the RCPSP benchmark data sets used are characterized in Section 7.2.2. By the way of contrast, no challenging data sets have yet been introduced for GRCPSP. Therefore, new ones have been generated as described in Section 7.2.3.

7.2.1 Complexity Measures

Comprehensive computational experiments can be found in the literature comparing and evaluating the efficiency of lower bound arguments as well as heuristic and exact procedures for solving RCPSP. For exact procedures, they all show that even though RCPSP is NP-hard, a large number of instances considered can be solved to optimality very quickly, whereas others remain computationally intractable for a restricted amount of computation time. For lower bound arguments and heuristic procedures, they reveal that for some instances bounds (solutions) close to the optimum can be found in short time, while large deviations have to be accepted for other ones.

Hence, a large number of different complexity measures have been proposed in the literature trying to predict the complexity of an instance in terms of the computation time required to achieve satisfying results. Providing such information may help to decide which solution procedure should be applied, in particular, whether exact solution procedures can be used or heuristic procedures should be preferred. Furthermore, different configurations of a procedure may perform well for different problem instances. If the instances for which certain configurations yield the best results can be described by values of complexity measures, the procedure can be adapted such that it automatically chooses the preferable configuration. Typical examples of such procedures are the adaptive sampling procedures discussed in Section 5.2.4, pp. 187.

In this context, it is important to state that according measures can not exactly predict the computational complexity of an instance, but may help to reveal basic tendencies. Furthermore, most of the statements can only refer to a certain type of solution procedure. For example, having available a restricted amount of computation time, one heuristic may outperform another one for a certain type of instances and vice versa.

In the following, the main characteristics of RCPSP as well as their assessed influences on the expected complexity of problem instances are briefly discussed. In particular, these are the number of jobs, the precedence relationships, and the resource constraints. Furthermore, a selection of appropriate complexity measures is described which are most frequently used in the literature for examining the efficiency of procedures.

Number of Jobs. As discussed in the Chapters 5 and 6, an optimal solution for RCPSP can, e.g., be determined by examining all precedence feasible sequences of jobs and constructing corresponding feasible schedules using the serial scheduling scheme. In case of discarding all precedence constraints, $n!$ sequences have to be evaluated. That is, with an increasing number of jobs the complexity is expected to grow exponentially depending on the restrictiveness of the precedence relationships.

Precedence Constraints. Basically, the number of feasible sequences is reduced when additional precedence constraints are introduced. This may lead to the assumption, that the problem complexity decreases with an increasing number of precedence relationships. However, the existence of precedence relationships may also result in the contrary effect. Finding an optimal solution may be-

come more difficult, because the possibilities of scheduling available jobs may be restricted in a disadvantageous manner by scheduling decisions made earlier. For example, consider a pair of jobs (h,j) which have to be processed in parallel within an optimal solution, because jobs h and j are incompatible to all other jobs, respectively. Furthermore assume that job h has to be completed at the end of period t the latest. Whenever (transitive) predecessors of job j are scheduled such that it can not start before period $t+1$, the resulting solution may be rather poor, because the jobs h and j have to be processed exclusively.

For RCPSP, the most popular complexity measure concerning precedence relationships is the *network complexity* NC which is defined as the ratio of non-redundant precedence relations to the number of jobs (cf. Definition 3.5, p. 76). This measure has initially been introduced by Pascoe (1966) and, among others, is used by Davis (1975) and Kolisch et al. (1995). Furthermore, it represents one of the control parameters employed within PROGEN (cf. Section 7.2.2). Nevertheless, it has the disadvantage that it only refers to the number of precedence relationships and, hence, reflects the structure of a project network only partially.

Further measures are the *complexity index* defined by De Reyck and Herroelen (1996) and the *restrictiveness* introduced by Thesen (1977). Since both measures can not be described briefly and are not used for evaluation purposes within our computational experiments, we omit giving respective descriptions. For detailed discussions referring to these measures see also De Reyck (1995) as well as Schwindt (1995, 1996).

Resource Constraints. According to the number of jobs, instances tend to become more complex when an increasing number of resource types have to be considered. Basically, this is due to the circumstance that finding combinations of jobs which in case of parallel execution efficiently exploit the available resources is more challenging. The same is true, if the average portion of resource types required for the execution of each job increases. This observation has led to the development of the *resource factor* RF which has initially been introduced by Pascoe (1966). Considering only the non-dummy jobs, it can be calculated as follows:

$$RF = \frac{1}{m} \cdot \frac{1}{(n-2)} \cdot \sum_{j=2}^{n-1} \sum_{r=1}^m \left\{ \begin{array}{ll} 1, & \text{if } u_{jr} > 0 \\ 0, & \text{otherwise} \end{array} \right\} \quad (7.1)$$

In case of $RF = 1$, processing any job requires all resource types. Vice versa, with $RF = 0$, no resource constraints have to be observed. Besides representing a control parameter of PROGEN, the resource factor has, e.g., been used in computational studies performed by Cooper (1976) and Alvarez-Valdés and Tamarit (1989 b).

However, the resource factor does not reflect whether the restricted resource availability imposes a constraint at all. For example, the total resource demand of each job combination which can be processed concurrently without violating the precedence constraints may not exceed the availability of any resource type. Then, an optimal solution with a project completion time equal to the length LBC1 of the critical path is defined by an *earliest starting time schedule*. This schedule is obtained by scheduling all jobs to begin at these earliest starting times yielded by a forward pass (cf. Section 2.2.3, pp. 49).

For this reason, a further complexity measure, called *resource strength* RS, has been proposed by Kolisch et al. (1995) which provides an extension of a measure initially introduced by Cooper (1976) and is used as a control parameter for PROGEN. For each resource type r , it considers the per period availability a_r , the maximum demand of a single job $\kappa_r^{\max} = \max\{u_{jr} \mid j = 2, \dots, n-1\}$ as well as the peak demand κ_r^{peak} which arises in any of the periods until the project completion when the earliest starting time schedule is realized. Then, the resource strength RS_r of resource type r can be computed as follows:

$$RS_r = \frac{(a_r - \kappa_r^{\max})}{(\kappa_r^{\text{peak}} - \kappa_r^{\max})} \quad (7.2)$$

RS denotes the average value $\frac{1}{m} \cdot \sum_{r=1}^m RS_r$ obtained for all resource types. If $a_r < \kappa_r^{\max}$ ($RS_r < 0$) for any resource type r , there obviously does not exist a feasible schedule, whereas in case of $a_r \geq \kappa_r^{\text{peak}}$ ($RS_r \geq 1$) for all resource types $r = 1, \dots, m$ the earliest starting time schedule represents an optimal solution. In general, the complexity of an instance tends to rise when the availability of a resource type is scarce compared to its demand. This is the case for a small resource strength.

Example. In our RCPSP example defined by Figure 2.2, p. 39, and Table 2.4, p. 51, we have $n = 12$ jobs and 16 non-redundant arcs. Hence, the network complexity is $NC = 16/12 = 1.33$. Since each non-dummy job uses the single re-

source type $r = 1$, the resource factor is $RF = 1$. For the earliest starting time schedule, the peak demand of $\kappa_1^{\text{peak}} = 8$ occurs in period 1 in which the jobs 2, 3, and 4 are processed simultaneously (cf. Figure 2.6, p. 52). Due to $a_1 = 4$ and $\kappa_1^{\text{max}} = \kappa_{31} = \kappa_{41} = \kappa_{81} = 3$, we yield $RS_1 = (4 - 3)/(8 - 3) = 1/5 = 0.2$.

In the following, we restrict to using the complexity measures NC, RF, and RS because they have been widely accepted in the literature. Besides the complexity measures described above, further ones are, e.g., presented in Davis (1975), Patterson (1976), Elmaghraby and Herroelen (1980), and Thomas and Salhi (1997). For GRCPSP, no specialized complexity measures are on hand. However, all of the measures presented above can immediately be transferred except for the resource strength. This is due to the varying resource availabilities in case of GRCPSP.

7.2.2 Data Sets for RCPSP

As described earlier, comprehensive computational experiments concerning RCPSP procedures have been reported in the literature. Most of these experiments (including those in this book) refer to one of the standard benchmark data sets PATT, SMFF, and J60. The first of these data sets (PATT) has been collected from the literature by Patterson (1984). Since today all its instances can be solved very efficiently by modern branch and bound procedures, Kolisch et al. (1995) have developed the problem generator PROGEN in order to create more challenging instances. With this generator, the new data sets SMFF (J30) and J60 have been generated serving as reference sets in most of the recent publications (cf., e.g., Kolisch and Sprecher (1996) and Kolisch et al. (1999)).

The generation of further data sets is, e.g., described in Alvarez-Valdés and Tamarit (1989 b) and Boctor (1990). However, since the corresponding data sets have not been established in the literature, we do not consider them within our experimental investigations. Besides PROGEN, further problem generators restricting to the generation of project networks have been developed by Demeuemeester et al. (1993) and Agrawal et al. (1996). Different extensions of PROGEN to RCPSP with maximum time lags are presented in Schwindt (1995, 1996, 1998b).

In the following, we describe the data sets PATT, SMFF, and J60, which are considered within our tests, in more detail. Table 7.1 summarizes the main

characteristics and complexity measures recorded for these data sets. "# inst." denotes the number of instances contained in each set.

Table 7.1. Characteristics of data sets

data set	# inst.	n	m	NC	RF	RS
PATT	110	7 - 51	1 - 3	1.08 - 3.05	0.25 - 1	0.00 - 1.42
SMFF (J30)	480 (360)	32	4	1.5, 1.8, 2.1	0.25, 0.5, 0.75, 1	0.2, 0.5, 0.7, 1
J60	480 (360)	62	4	1.5, 1.8, 2.1	0.25, 0.5, 0.75, 1	0.2, 0.5, 0.7, 1

Data Set PATT. This data set, which has been collected from the literature, contains instances originally described in Davis (1969), Patterson and Huber (1974), and Talbot and Patterson (1978). Some of these instances correspond to practical problems whereas others have been generated randomly. As a consequence, every instance shows different values for the different problem characteristics and complexity measures. Therefore, a range of the values obtained is given in each column, with RS containing the average value of all resource types. For three instances, RS_r is larger than 1 for all resource types and, hence, an optimal solution is easily determined by the earliest starting time schedule. That is, for these instances the different bound arguments as well as the destructive improvement technique may not improve on the critical path lower bound LBC1. Nevertheless, these instances are considered for the sake of completeness.

Data Sets SMFF and J60. These data sets have systematically been generated by Kolisch et al. (1995) using the problem generator PROGEN. For this purpose, the generator can be configured by specifying values for a set of control parameters which have to be met by the randomly created instances. Besides the number of jobs and the number of resource types, this includes the complexity measures NC, RF, and RS. For all resource types, RS_r possesses the same value. Further, less important control parameters, the definitions of which are not reported here, refer, e.g., to the minimum and maximum number of predecessors and successors allowed for each job (cf. Kolisch et al. (1995) as well as Kolisch and Sprecher (1996) for details).

In case of SMFF (J30), instances with 32 (30 non-dummy) jobs and 4 resource types have been generated. Furthermore, the values for NC, RF, and RS given in Table 7.1 have systematically been combined resulting in $3 \cdot 4 \cdot 4 = 48$ differ-

ent combinations. For each of these combinations, a group of ten instances has been generated such that a total of 480 instances is obtained. However, the resulting data set contains 120 instances with a resource strength of $RS_r = 1.0$ for all resource types. Since for these instance optimal solutions can easily be computed, the 12 corresponding groups are excluded within our tests.

The data set J60 has been generated according to SMFF except for the difference that the instances consist of 60 non-dummy jobs. Again, those 12 problem groups with $RS = 1.0$ are discarded within our computational experiments.

7.2.3 Data Sets for GRCPSP

For GRCPSP, only an adapted version of the data set PATT is available which has been introduced in Simpson and Patterson (1996). Unfortunately, this data set considers only precedence relationships of the conventional finish-to-start type. Furthermore, the instances contained are not challenging except for one (cf. Demeulemeester and Herroelen (1997 a)). Therefore, we have generated new data sets based on the instances contained in SMFF. This data set is preferred to PATT and J60 for the following reasons. In contrast to PATT, it has been generated systematically which allows for a more sound analysis of the computational experiments. For J60, many instances can not be solved to optimality within reasonable time even applying the most efficient exact RCPSP procedures available. Since GRCPSP represents the more general problem, the instances obtained from the J60 data set would be even more challenging. However, in particular for the comparison of heuristic procedures, it is advantageous to have optimal solutions for the instances considered on hand.

When generating GRCPSP instances from RCPSP ones, the following steps have to be performed. For each instance, minimum time lags have to be introduced, dynamic availabilities of the resource types have to be determined, and time windows for the jobs have to be defined. Systematically varying the generation process for each step results in different types of *start-to-start project networks*, *resource availability profiles*, and *sets of time windows* which, subsequently, can arbitrarily be combined in order to obtain the GRCPSP data sets. This circumstance is schematically depicted in Figure 7.1. The figure shows that only two types of project networks (P_0, P_1) and time windows (T_0, T_1) are generated whereas a total of six different types of resource profiles (R_0 to R_4 , RSP) are considered. This is due to the computational experiments revealing that the type of resource profiles to be considered influences the computational

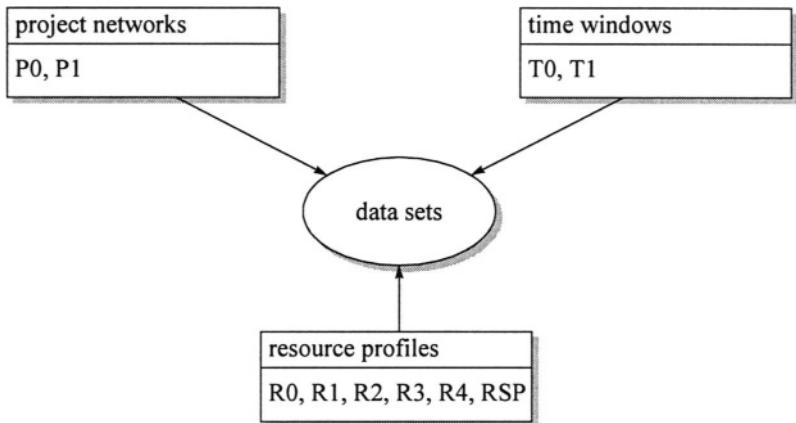


Figure 7.1. Generation of data sets

complexity of an instance much stronger than the types of precedence relationships or time windows. This observation is also confirmed by the computational results presented in Demeulemeester and Herroelen (1997 a).

Due to the chosen implementation of the generation processes two basic classes of data sets can be distinguished. The instances of the first class are obtained by taking an optimal solution of an RCPSP instance and generating project networks, resource profiles, and time windows such that the RCPSP solution remains optimal for GRCPSP. This has the advantage that the optimal objective function values of the GRCPSP instances are known in advance and that minimum time lags and time windows can be chosen such that feasible solutions exist for all instances. A second, more challenging class with the optimal solutions being different from the ones of the underlying RCPSP instances is yielded by constructing resource profiles using a dynamic version of the resource strength as a control parameter. Note that those 12 SMFF groups (120 instances) with a resource strength of RS = 1.0 are completely omitted for generating GRCPSP data sets.

Minimum Time Lags. For a first type of project networks **P0**, each finish-to-start precedence relationship between a job i and a job j defined in the original SMFF instance is converted into a start-to-start precedence relationship with a minimum time lag $\lambda_{ij} = d_i$. That is, the GRCPSP instances of the type P0 contain only conventional finish-to-start relationships.

The second type **P1** extends the project networks of P0 by randomly introducing minimum time lags of the types $\lambda_{ij} > d_i$ and $\lambda_{ij} < d_i$, respectively, such that the resulting start-to-start project network remains non-cyclical. The number of minimum time lags of each type is randomly chosen from the interval $[0, 0.1 \cdot DP]$ with DP denoting the number of non-redundant precedence relations contained in the original SMFF project network. Note that in order to obtain instances for the first class the values λ_{ij} are chosen such that the starting times of the optimal SMFF solution remain feasible for the resulting start-to-start project network.

Example. Consider our RCPSP example defined by Figure 2.2, p. 39, and Table 2.4, p. 51. An optimal solution for this instance is given in Figure 7.2. Additional precedence relationships could, e.g., be introduced between the job pairs (2, 6) and (7, 9). For the pair (2, 6), a minimum time lag of the type $\lambda_{ij} > d_i$ can be introduced by choosing values $\lambda_{26} \in [4, 9]$ with the optimal schedule remaining feasible. A minimum time lag of the type $\lambda_{ij} < d_i$ for the pair (7, 9) may be defined by $\lambda_{79} \in [0, 2]$.

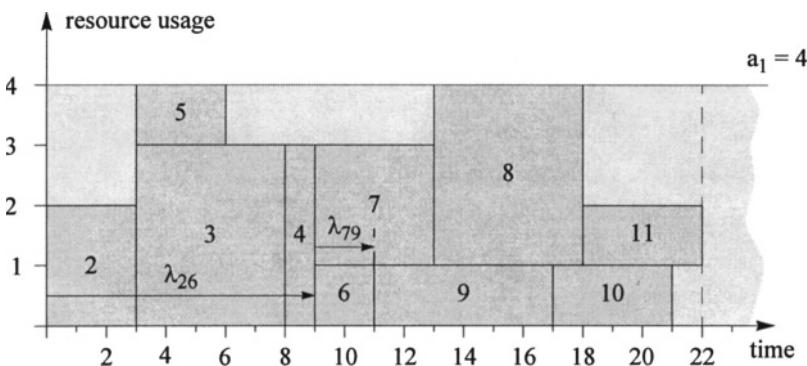


Figure 7.2. Optimal RCPSP schedule

Resource Availability. Different types of resource profiles are generated as follows. For each instance, an optimal RCPSP solution is considered with CT denoting the project duration. Then, for all periods $t = 1, \dots, CT$ and for each resource type $r = 1, \dots, m$, the capacity requirement κ_{rt} can be computed which is utilized by the jobs being active in period t. A first type of resource profiles **R0** is obtained by limiting the availability of resource type r in the periods $t = 1, \dots, CT$ to $a_{rt} := \kappa_{rt}$. To have a planning horizon \bar{T} larger than CT, the resource profile is replicated three times resulting in $\bar{T} = 4 \cdot CT$.

However, instances with a resource profile of type R0 may not be very challenging for exact solution procedures (cf. Section 7.5). In particular, for optimal solutions of SMFF instances with a small resource factor, the jobs active in a certain period often use only a small subset of the resource types considered. For all other resource types not required in this period the availabilities in the generated resource profiles are set to zero. As a consequence, within PROGRESS and SCATTER, only a subset of all available jobs may be eligible at a given decision point such that the number of subproblems to be developed in each node is small. Therefore, further (more challenging) types of resource profiles are constructed by defining a lower bound $\gamma \cdot a_r$ on the minimum capacity which has to be provided for each resource type r in any period (a_r denotes the availability of resource type r in the SMFF instance). That is, the availability of a resource type r in a period t is determined by $a_{rt} = \max\{\kappa_{rt}, \gamma \cdot a_r\}$. Choosing the values $\gamma = 0.25, 0.5, 0.75$, and 1 results in resource profiles of the type **R1**, **R2**, **R3**, and **R4**, respectively. Note that the data set obtained by combining project networks of type P0 with resource profiles of type R4 (constant resource availability) is equal to SMFF, if time windows are omitted (T_0 , see below).

Example. Consider the optimal solution with $CT = 22$ for our RCPSP example given in Figure 7.2. The resulting resource profile for $r = 1$ and the periods $t = 1, \dots, 22$ is depicted in Figure 7.3. In order to yield an extended planning horizon of $\bar{T} = CT \cdot 4 = 88$, the profile is replicated in the periods $23, \dots, 44$, $45, \dots, 66$, and $67, \dots, 88$ thereby defining the type R0. For determining a profile of the type R3, at least $0.75 \cdot a_r = 3$ capacity units have to be provided in each period. Therefore, the availability of R0 has, e.g., to be increased in the period intervals $[1, 3]$ and $[19, 25]$.

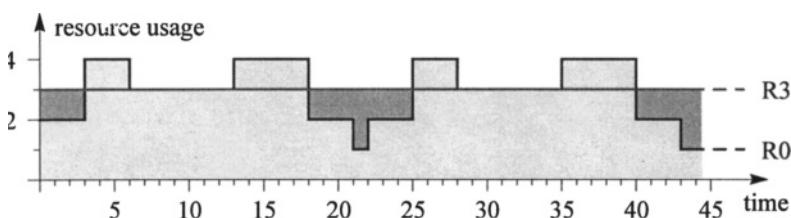


Figure 7.3. Generation of resource profiles R0 and R3

As the computational results in the following sections show, instances of the data sets obtained for resource profiles of the types R0 to R3 are still easier to

solve than those yielded for R4 (at least for exact procedures). This might lead to the conclusion that the computational complexity of instances increases if the corresponding resource profile becomes more balanced. To prove the opposite and to show that GRCPSP represents a more challenging optimization problem than its special case RCPSP, a last type **RSP** of resource profiles is generated using the resource strengths of the respective SMFF instances as control parameters.

For each SMFF instance, the resource constraints are discarded and a solution is constructed by starting all jobs $j = 1, \dots, n$ at their earliest starting times obtained by a simple forward pass. Again, $CT = LBC1$ denotes the corresponding project completion time and κ_{rt} defines the capacity requirement of resource type r for processing all jobs being active in period t . With $\kappa_r^{\max} = \max\{u_{jr} \mid j = 2, \dots, n-1\}$ denoting the maximum resource demand of a single job, the availability of resource type r in period $t = 1, \dots, CT$ is set to $a_{rt} = \kappa_r^{\max} + \lfloor RS \cdot (\kappa_{rt} - \kappa_r^{\max}) \rfloor$ depending on the resource strength RS (cf. Section 7.2.1). In order to guarantee that a feasible solution exists for a corresponding GRCPSP instance, the planning horizon \bar{T} is set to $4 \cdot CT$ replicating the resource profile for three times. Note that the optimal objective function values of instances considering resource profiles of type RSP are not equal to the ones of the original SMFF instances (T_0 , see below).

Example. If the resource constraints are omitted in our RCPSP example, the smallest possible project completion is $CT = 16$. Realizing the earliest starting time schedule, the demand of resource type 1 in period 1 is $\kappa_{11} = 8$ (cf. Figure 2.6, p. 52). For $RS = 0.2$, we yield $a_{11} = 3 + \lfloor 0.2 \cdot (8-3) \rfloor = 3 + 1 = 4$. For the remaining periods $t = 2, \dots, 16$ with $\kappa_{1t} < 8$, $a_{1t} = 3$ is obtained. Subsequently, this resource profile is replicated 3 times resulting in a planning horizon of $\bar{T} = 64$.

Remark 7.1. Due to the generation process used for constructing resource profiles of the type RSP, the minimum availability of a resource type r occurring in any period of the planning horizon \bar{T} is not smaller than the maximum demand of any job. That is, each job can be executed in all periods of the planning horizon if the resource constraints are omitted. By the way of contrast, this is not guaranteed for the resource profiles of the types R0 to R3. Within the corresponding data sets, a job j might be processed only in a few period intervals of the planning horizon, because not enough resources are available in the remaining periods.

Time Windows. As **T0** we denote the case of time windows being ignored. A type **T1** is obtained as follows. For each underlying SMFF instance, the number of jobs with time windows is randomly chosen from the interval $[0, 0.1 \cdot n]$. Subsequently, the time windows are defined for randomly selected jobs such that the underlying optimal SMFF solution remains feasible.

Example. Consider the optimal schedule given in Figure 7.2 with a project completion time of $CT = 22$. Performing a forward and backward pass (initialized with $LF_n = LS_n = 22$), yields the time window $[5, 13]$ for job 6 which is started at $SS_6 = 9$ and finished at $SF_6 = 11$. Hence, feasible release and due dates may, e.g., be chosen from the intervals $[5, 9]$ and $[11, 13]$, respectively.

Selected Data Sets. As already mentioned, different data sets are obtained by combining the project networks P0 or P1 with resource profiles R0 to R4 or RSP. Having the option of ignoring (T0) or observing (T1) time windows, this results in a total of 24 *different data sets* with 360 *instances* each. The data sets are denoted by $Px/Ry/Tz$. For example, P1/R3/T1 represents the data set which is obtained by combining the start-to-start project networks P1 with the resource profiles of the type R3. Furthermore, time windows of the type T1 are considered.

However, in order to reduce the computational effort required for analyzing the performance of the different procedures, we restrict ourselves to those data sets which are most challenging and which have provided the most interesting results in preliminary computational experiments. They consist of the project networks P0 combined with all resource profiles (R0 to R4, RSP) as well as the project networks P1 together with the resource profiles R3, R4 and RSP. Time windows of type T1 are excluded except for the single data set P1/R3/T1. This is due to the fact that for exact procedures the corresponding instances are easier to solve than the ones without time window restrictions. Furthermore, heuristic procedures, in particular priority-rule based heuristics, may fail in determining feasible solutions for some instances of this data set.

7.3 Lower Bound Arguments

In this section, we report on results of comprehensive numerical experiments regarding the effectiveness of lower bound arguments based on the RCPSP standard benchmark data sets PATT, SMFF, and J60. After comparing the effec-

tiveness of simple bounding procedures, the approach of destructive improvement is examined thoroughly by evaluating different combinations of reduction techniques and bound arguments. Further computational results are contained in Klein and Scholl (1999). By the way of contrast, we renounce presenting results for GRCPSP. Preliminary computational results have shown that the conclusions which can be drawn from the RCPSP results are also valid for GRCPSP. Therefore, providing such results does not give additional insight and is omitted.

7.3.1 Simple Bound Arguments

In the sequel, the effectiveness of "old" and "new" simple bound arguments is examined. As new ones, we consider all those arguments which have been introduced by the author (cf. Table 7.3). The remaining (old) ones have been presented in the literature before (cf. Table 7.2). The bound arguments are compared with respect to the following measures (CT is the optimal or best known objective function value):

- av.dev: average relative deviation of the bound LB_x from CT in %
(relative deviation: $(CT - LB_x)/CT \cdot 100\%$)
- max.dev: maximum relative deviation of LB_x from CT in %

For PATT and SMFF, the optimal solutions to all instances are known. In case of J60, the solutions yielded by the best configuration of SCATTER after 1 hour of computation time are used (cf. Section 7.5.2.2, pp. 318). Computation times required by the different bound arguments are omitted, because they are negligibly small for all instances.

Table 7.2. Effectiveness of existing lower bound arguments

data set		LBC1	LBC2	LBC3	LBN1	LBM1
PATT	av.dev	13.53	24.17	10.73	11.32	10.62
	max.dev	51.56	57.89	43.75	36.00	32.81
SMFF	av.dev	12.28	38.50	8.87	8.19	8.76
	max.dev	54.72	68.85	45.28	36.21	32.22
J60	av.dev	10.09	35.30	8.99	10.82	7.16
	max.dev	52.76	71.08	49.60	51.30	39.05

Table 7.2 and Table 7.3 show the results obtained for existing and new bound arguments, respectively. The shaded cells mark the procedures performing best

for each data set and class of bound arguments. LBM1 has been computed for all resource types and up to five "machines". For LBN1, LBN2, and LBN3, different sets S are obtained by rotating the priority list as described for LBN1.

Among the existing bound arguments, the most recently introduced ones, LBN1 and LBM1 come out first. Comparing LBN1 and LBN2 reveals that the extensions included are very successful. For J60, the average deviation is reduced by more than 33%. LBM1 is clearly outperformed by the arguments which are based on incompatible pairs and triplets (LBM2, LBM3) and consider all resources for the data sets PATT and SMFF (cf. the shaded cells). This indicates that discarding all but one resource is often a too strong relaxation. This observation is confirmed by the results obtained for the bin packing bounds LBB1 and LBB2, which also show large deviations. Note that the maximum deviation of 100% obtained by LBB1 for J60 is due to the circumstance that for some instances all jobs possess a per period resource usage smaller than one third of the availability for all resource types and, hence, $LBB1 = 0$.

Table 7.3. Effectiveness of the new lower bound arguments

data set		LBB1	LBB2	LBN2	LBN3	LBM2	LBM3	LBP1	LBP2
PATT	av.dev	37.94	23.31	7.43	5.89	7.78	5.87	11.16	8.63
	max.dev	70.37	57.89	24.00	25.00	28.00	25.00	43.75	39.06
SMFF	av.dev	45.20	35.79	6.05	6.48	5.28	7.00	9.69	8.98
	max.dev	100.00	68.85	26.98	23.88	34.33	25.86	49.60	44.34
J60	av.dev	59.64	34.32	6.85	6.97	8.04	8.91	9.14	8.87
	max.dev	100.00	71.07	31.67	31.67	49.60	49.60	49.60	47.85

In contrast to the data sets PATT and SMFF, LBM1 clearly yields better results than its extensions LBM2 and LBM3 for J60. This can be explained by the fact that the number of incompatible pairs and triplets is rather small for instances of this data set. On average, 26.19 pairs of jobs of an SMFF instance are incompatible due to the resource constraints. For J60, though containing twice as much jobs, the average number of resource incompatible pairs is only 36.6. For the number of incompatible triplets approximately the same relationship holds. Therefore, LBM1 may outperform LBM2 and LBM3 for this data set, because it also considers incompatibilities among 4 and 5 jobs, respectively.

The success of the procedures LBN3 and LBM3 shows that the increase in the computational complexity, caused by checking for triplets, pays off. Comparing LBN2, LBN3 and their relatives LBM2, LBM3 indicates that different strate-

gies should be applied in order to find incompatible subsets. Despite the simplicity of their approaches, the precedence based bound arguments LBP1 and LBP2 perform nearly as good as the old bounds, in particular for PATT.

Table 7.4. Comparison of old and new lower bound arguments

data set		LB^{old}	LB^{new}	LB^*
PATT	av.dev	7.62	3.74	3.74
	max.dev	28.00	13.04	13.04
SMFF	av.dev	5.41	4.05	4.05
	max.dev	26.09	19.74	19.74
J60	av.dev	5.65	5.29	5.29
	max.dev	27.97	27.97	27.97

Comparing the best bound values obtained by existing and new procedures (columns LB^{old} and LB^{new} in Table 7.4), respectively, shows that the new bound arguments clearly outperform the existing ones. For PATT, the average as well as the maximum deviations are more than halved. The superiority is underlined by comparing LB^{new} with the maximum bound value LB^* yielded by all bound arguments. That is, the existing bound arguments can be omitted without any loss in bound quality.

Remark 7.2. Partially, the superiority of the new (and extended) bound arguments can be explained by their way of construction. For example, the arguments LBC3, LBP1, and LBP2 start from computing LBC1 such that they dominate this basic bound. Furthermore, the bound LBN1 and its derivatives LBN2 and LBM2 dominate LBC1 if the set S generated contains the jobs of the critical path. Since LBN2 directly extends LBN1 there is also a dominance relationship when the same sorting criterion is used. Furthermore, LBM1 dominates LBC2 as indicated in Remark 4.3, p. 126.

Some of the bound arguments show a poor performance considering the measures av.dev and max.dev. So, it has to be examined if their application can be justified at all. Table 7.5 gives the quota of instances of each data set, for which LBx is one of the arguments yielding the maximum bound value. It reveals that some of the bound arguments showing large deviations surprisingly obtain the best bound for a large portion of the instances, in particular, LBC3, LBP1, and LBP2. This indicates that their success in yielding strong bounds strongly depends on the structure of a problem instance considered.

Table 7.5. Quotas of maximum bound values

LBx	LBC1	LBC2	LBC3	LBN1	LBM1
PATT	29.09	1.82	32.73	24.55	32.73
SMFF	43.61	1.67	60.00	43.89	49.72
J60	61.39	14.17	68.06	43.33	68.61

LBx	LBB1	LBB2	LBN2	LBN3	LBM2	LBM3	LBP1	LBP2
PATT	3.64	4.55	44.55	62.73	43.64	68.18	30.91	50.00
SMFF	0.83	2.22	59.44	53.61	75.56	51.94	58.06	59.17
J60	0.00	14.17	63.61	63.06	73.61	63.06	68.89	68.89

Furthermore, the bound arguments based on discarding all but one resource type, in particular, LBC2 and LBM1, clearly yield better results for J60 than for PATT and SMFF. This due to the instances of J60 containing a rather small number of incompatible pairs and triplets as stated above. As a consequence, the bound arguments which do not rely on such incompatibilities become more effective. Nevertheless, in case of J60, LBM2 obtains the best bound value for more instances than any other bound argument, though concerning the measures av.dev and max.dev it performs rather poor (cf. Table 7.3).

7.3.2 Destructive Improvement

In the following, we examine the effectiveness of the destructive improvement approach by incorporating different reduction techniques and bounding procedures and applying the resulting procedures to the benchmark data sets. The combinations of the reduction techniques considered are given in Table 7.6. Note that reduction by forward and backward pass is always included.

Table 7.6. Combinations of reduction techniques

Reduction by	RS	RP	RC	RPC	RSP	RSC	RSPC
subprojects	■				■	■	■
precedence relations		■		■	■		■
core times			■	■		■	■

If reduction by subprojects is employed, the lower bound arguments and the destructive improvement technique are applied to the initial project as well as to all subprojects. Since the computational effort per instance strongly depends on

the chosen combination and is partially remarkably large, the average computation time (av.cpu) in seconds is introduced in addition to the measures used in the previous section.

Pure Reduction Process. In order to compare the potential of the different reduction techniques, they have been applied in the destructive improvement process without using any lower bound arguments except for LBC1. That is, only the reduction step is performed (cf. Figure 4.4, p. 138). The results are given in Table 7.7. The combination RS has been omitted from this test, because it does not improve on LBC1 until any other bound argument or reduction technique is applied. The settings RP and RC perform nearly identical concerning av.dev, whereas for max.dev RP is slightly better than RC in case of PATT and SMFF but requires considerably larger computation times, in particular for J60. Nevertheless, each technique yields smaller values for av.dev than the best existing bounds LB^{old} except for SMFF. Combining them to RPC leads to a further decrease of av.dev. This combination obtains already better average deviations than LB* for J60 though its computation takes less time.

Table 7.7. Destructive improvement without bound arguments

data set		LB ^{old}	LB*	RP	RC	RPC	RSP	RSC	RSPC
PATT	av.dev	7.62	3.74	6.37	6.21	5.70	4.06	3.94	3.16
	max.dev	28.00	13.04	32.00	38.00	32.00	23.44	26.56	23.44
	av.cpu	0.01	0.05	0.02	0.01	0.02	0.22	0.05	0.24
SMFF	av.dev	5.41	4.05	5.92	6.14	5.54	4.24	4.50	3.81
	max.dev	26.09	19.74	32.80	33.71	32.80	29.35	31.25	29.35
	av.cpu	0.01	0.06	0.03	0.01	0.03	0.16	0.05	0.17
J60	av.dev	5.65	5.29	5.28	5.33	5.16	4.77	4.82	4.63
	max.dev	27.97	27.97	27.35	27.35	27.35	25.64	25.64	25.64
	av.cpu	0.03	0.33	0.20	0.04	0.21	1.43	0.34	1.48

If subprojects are evaluated the bound quality is considerably improved in particular for PATT and SMFF. However, the average computation times are increased, too. This is particularly true, if subprojects as well as additional precedence relations are examined. Using all reduction techniques clearly obtains the best results, indicating that they complement each other well. RSPC outperforms LB* for all data sets concerning av.dev. In general, the bound arguments lead to smaller maximum deviations, because the large variety of approaches is successful even for special structures of problem instances.

Destructive Improvement With All Bound Arguments. In order to examine the full potential of the destructive improvement technique, the same test is repeated additionally using all bounding procedures on hand. The reduction process is started with the best bound value LB*. The results in Table 7.8 show that if only one reduction technique is applied, RC is most successful for all data sets. It obtains the smallest deviations taking the shortest computation times. By the way of contrast, RS takes much more computation time but hardly improves on LB*. If two reduction techniques are combined, RPC is the fastest strategy, but it yields results clearly worse than those of RSP and RSC. Evaluating subprojects by applying all bound arguments and destructive improvement is computationally very expensive. Nevertheless, the results for RSPC underline the power of the destructive improvement technique. The values of av.dev are nearly halved compared to those of LB* for the data sets PATT and SMFF.

Table 7.8. Destructive improvement with all bound arguments

data set		LB ^{old}	LB*	RS	RP	RC	RPC	RSP	RSC	RSPC
PATT	av.dev	7.62	3.74	3.43	3.10	3.03	2.89	2.28	2.14	1.97
	max.dev	28.00	13.04	13.04	13.04	13.04	13.04	13.04	13.04	13.04
	av.cpu	0.01	0.05	0.79	0.12	0.11	0.12	1.88	1.72	1.91
SMFF	av.dev	5.41	4.05	3.99	3.25	3.06	2.88	2.64	2.48	2.25
	max.dev	26.09	19.74	19.74	19.74	19.74	19.74	19.74	19.74	19.74
	av.cpu	0.01	0.06	0.44	0.15	0.12	0.15	1.07	0.96	1.08
J60	av.dev	5.65	5.29	5.21	4.85	4.84	4.72	4.43	4.39	4.30
	max.dev	27.97	27.97	27.97	25.96	25.96	25.96	25.00	25.00	25.00
	av.cpu	0.03	0.33	3.56	0.88	0.73	0.89	9.11	8.07	9.19

Restricted Application Of Bounds. In order to reduce computational effort required by the bound computation, it seems to be appropriate to employ only a subset of the bounding procedures within the destructive improvement process. The subset which in preliminary experiments has turned out to be most successful in relation to computation times contains the arguments LBC2, LBM2, and LBM3 (cf. Table 7.5). Though LBC2 yields rather poor results if computed solely, its application is worthwhile within the destructive improvement technique. This is due to the circumstance that it is applied to time intervals for the reduced problem such that it can benefit from reductions made so far (cf. Section 4.2.2.2, pp. 147). Table 7.9 shows that restricting the initial bound computation and the reduction process to this subset reduces the computation times to

approximately one third. In general, the rather small loss in bound quality is clearly over-compensated by the gain in speed, in particular for J60. For example, the fastest combination RC can be computed faster than LB* but yields smaller average deviations for all data sets.

Table 7.9. Destructive improvement with LBC2, LBM2, LBM3

data set		LB ^{old}	LB*	RS	RP	RC	RPC	RSP	RSC	RSPC
PATT	av.dev	7.62	3.74	4.51	3.31	3.19	3.00	2.38	2.21	2.08
	max.dev	28.00	13.04	21.74	15.38	13.04	13.04	13.04	13.04	13.04
	av.cpu	0.01	0.05	0.18	0.04	0.03	0.05	0.61	0.44	0.63
SMFF	av.dev	5.41	4.05	4.61	3.37	3.18	2.99	2.73	2.56	2.34
	max.dev	26.09	19.74	19.74	19.74	19.74	19.74	19.74	19.74	19.74
	av.cpu	0.01	0.06	0.10	0.06	0.04	0.06	0.37	0.26	0.38
J60	av.dev	5.65	5.29	5.54	4.90	4.92	4.77	4.48	4.43	4.34
	max.dev	27.97	27.97	27.97	27.35	27.35	27.35	25.64	25.64	25.64
	av.cpu	0.03	0.33	0.95	0.43	0.28	0.33	3.47	2.43	3.54

7.3.3 Influence of the Problem Structure

Most of the existing and new bound arguments are based on the incompatibility of pairs or triplets of jobs. For the data sets considered, the number of incompatible pairs and triplets strongly depends on the average resource strength of an instance. For example, the maximum number of incompatible pairs of an SMFF instance with RS = 0.2 is 151, whereas it is only 22 for RS = 0.7.

Therefore, it may be promising to select the bound arguments and the reduction techniques used according to, e.g., the resource strength of an instance. In order to find out which combination should be selected, the instances of each data set have been divided into three classes C1, C2, and C3 subject to their resource strengths. The respective classes contain the instances with $RS \in [0, 1/3]$, $RS \in [1/3, 2/3]$, and $RS \geq 2/3$. The influence of other measures of problem complexity such as NC and RF on the performance of bound arguments is much less significant than that of RS. Hence, it does not seem to be appropriate to select reduction techniques and bound arguments on the basis of those measures. For the data set PATT, the classes C1, C2, and C3 consist of 11, 86, and 13 instances, respectively. For SMFF and J60, they all contain 120 instances due to the instances being generated systematically (cf. Table 7.1, p. 268).

Table 7.10. Influence of the resource strength for PATT

PATT		destructive improvement								
		without arguments			with LBC2, LBM2, LBM3			with all arguments		
		RPC	RSC	RSPC	RPC	RSC	RSPC	RPC	RSC	RSPC
class	LB ^{old} LB*									
C1	av.dev	5.02	1.77	13.96	11.65	7.50	1.21	1.77	1.21	1.21
	max.dev	10.53	5.26	32.00	26.56	23.44	5.26	5.26	5.26	5.26
	av.cpu	0.01	0.03	0.01	0.02	0.08	0.02	0.09	0.14	0.06
C2	av.dev	8.49	4.35	5.56	3.74	3.18	3.48	2.59	2.47	3.35
	max.dev	28.00	13.04	21.88	15.62	13.04	13.04	13.04	13.04	13.04
	av.cpu	0.01	0.05	0.02	0.05	0.24	0.05	0.41	0.60	0.12
C3	av.dev	3.49	0.96	0.96	0.00	0.00	0.96	0.00	0.00	0.96
	max.dev	25.00	12.50	12.50	0.00	0.00	12.50	0.00	0.00	12.50
	av.cpu	0.01	0.8	0.02	0.08	0.37	0.07	0.91	1.21	0.19

Among the combinations of reduction techniques given in Table 7.6, RPC, RSC, and RSPC have been selected for the examination. The first two represent the most promising compromise concerning computation time and bound quality. The last one is chosen, because it uses the full potential of the destructive improvement technique. These combinations are applied without any bound arguments, together with LBC2, LBM2, and LBM3 as well as with all arguments, respectively. The results obtained can be found in the Tables 7.10 to 7.12. The

Table 7.11. Influence of the resource strength for SMFF

SMFF		destructive improvement								
		without arguments			with LBC2, LBM2, LBM3			with all arguments		
		RPC	RSC	RSPC	RPC	RSC	RSPC	RPC	RSC	RSPC
class	LB ^{old} LB*									
C1	av.dev	10.55	7.58	14.50	11.55	9.65	6.91	5.75	5.22	6.59
	max.dev	26.09	19.74	32.80	31.52	29.35	19.74	19.74	19.74	19.74
	av.cpu	0.01	0.07	0.05	0.06	0.25	0.07	0.27	0.45	0.17
C2	av.dev	4.27	3.52	1.77	1.60	1.47	1.71	1.60	1.47	1.68
	max.dev	18.87	15.09	11.32	11.32	11.32	11.32	11.32	11.32	11.32
	av.cpu	0.01	0.05	0.03	0.05	0.15	0.06	0.27	0.35	0.14
C3	av.dev	1.29	0.94	0.30	0.28	0.27	0.30	0.28	0.27	0.30
	max.dev	10.91	9.09	5.45	5.45	5.45	5.45	5.45	5.45	5.45
	av.cpu	0.01	0.05	0.02	0.05	0.12	0.06	0.28	0.33	0.13

darkly shaded cells mark those combinations giving the best bound quality, whereas the lightly shaded ones emphasize those combinations which yield a good bound quality with short computation times appearing to be most favorable.

Table 7.12. Influence of the resource strength for J60

J60		destructive improvement											
		without arguments				with LBC2, LBM2, LBM3				with all arguments			
class	LB ^{old}	LB*	RPC	RSC	RSPC	RPC	RSC	RSPC	RPC	RSC	RSPC		
C1	av.dev	16.35	15.63	14.23	13.24	12.69	13.09	12.08	11.82	12.93	11.94	11.70	
	max.dev	28.57	28.57	27.35	25.64	25.64	27.35	25.64	25.64	25.96	25.00	25.00	
	av.cpu	0.03	0.33	0.39	0.38	2.73	0.62	2.43	4.85	1.09	8.68	10.98	
C2	av.dev	2.78	2.55	1.21	1.18	1.18	1.21	1.18	1.18	1.20	1.18	1.18	
	max.dev	15.49	15.49	11.43	9.46	9.46	11.43	9.46	9.46	11.43	9.46	9.46	
	av.cpu	0.03	0.32	0.13	0.33	0.94	0.37	2.38	2.98	0.80	7.77	8.38	
C3	av.dev	0.24	0.15	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	
	max.dev	5.45	5.45	1.82	1.82	1.82	1.82	1.82	1.82	1.82	1.82	1.82	
	av.cpu	0.03	0.32	0.10	0.34	0.76	0.33	2.35	2.98	0.77	7.77	8.20	

In general, the results for PATT are less expressive than those for the other data sets, because it has not been generated systematically (cf. Table 7.10). For example, the maximum number of jobs of instances and the number of instances as well varies considerably for the different classes. Therefore, the following indications are mainly based on the new data sets (cf. Tables 7.11 and 7.12).

For the first class C1, RSPC combined with all bound arguments must be applied to obtain the best results. In order to avoid the high computation times, RPC with the restricted set of lower bound arguments (LBC2, LBM2, and LBM3) should be employed. Considering the classes C2 and C3, even those arguments can be excluded without loss in bound quality and, hence, should be omitted. It is remarkable that for SMFF and J60 pure RPC without any bound arguments always clearly outperforms LB* concerning the classes C2 and C3, though requiring the shorter computation times.

7.3.4 Comparison with Complex Bound Arguments

Finally, we compare the destructive improvement approach with the complex bound arguments presented in Section 4.1.2, in particular with the set covering based approach introduced by Mingozi et al. (1998).

Unfortunately, no (detailed) computational results are available in the literature for the LP-relaxation and Lagrangean relaxation based arguments concerning the data sets PATT, SMFF, and J60. Therefore, these concepts can not immediately be compared to destructive improvement. However, for the randomly generated instances examined by Christofides et al. (1987), computing lower bounds using the LP-relaxation with cutting planes yields slightly better results than using Lagrangean relaxation. Recently, Drexel and Kimms (1998) have studied the effectiveness of the Lagrangean relaxation based approach for the data sets SMFF and J60 without giving detailed results. They restrict to the statement that this argument is clearly outperformed by destructive improvement and the set covering based argument.

The effectiveness of the set covering based approach as described in Section 4.1.2 is examined by Mingozi et al. (1998) and Baar et al. (1998). In both papers, the results given indicate that this approach clearly outperforms the existing bound arguments of Table 7.2.

In the column MI of Table 7.13, the results for PATT and SMFF obtained by Mingozi et al. (1998) are shown which have been yielded on an IBM compatible personal computer with an 80386sx (15 MHz clockpulse) central processing unit using the general purpose solver XPress MP. Unfortunately, no computation times required for computing the corresponding bound values are provided. The results are compared to all simple bound arguments and destructive improvement. Note that the values for SMFF refer to a subset of 170 instances evaluated by Mingozi et al. (1998). The indices for the different destructive improvement configurations represent the bound arguments used. The index "ma" denotes that the capacity bound LBC2 as well as the machine bounds LBM2 and LBM3 are applied whereas the index "all" indicates that all simple bound arguments are computed.

Already RPC_{ma} obtains better results than the set covering based approach. Whereas the latter has proved to be computationally too expensive within a branch and bound procedure (cf. Mingozi et al. (1998)), this configuration of destructive improvement can be evaluated in an average time of less than 0.1

Table 7.13. Destructive improvement vs. set covering based approaches (SMFF)

data set		MI	LB*	RPC _{ma}	RPC _{all}	PSPC _{all}
PATT	av.dev	3.29	3.74	3.00	2.89	1.97
	max.dev	14.60	13.04	13.04	13.04	13.04
SMFF	av.dev	6.38	6.77	5.72	5.49	4.42
	max.dev	23.17	19.74	19.74	19.74	19.74

seconds. RSPC_{all} reduces the average deviation for PATT and SMFF by more than 40% and 30%, respectively.

In Baar et al. (1998), results are presented for a subset of SMFF instances which are (partially) included within those considered by Mingozi et al. (1998). Therefore, they are omitted. However, it is important to state that Baar et al. (1998) avoid determining all maximal compatible sets in advance by the concept of column generation thus providing a more efficient way of calculating the lower bound values.

Recently, Baar et al. (1998) have applied their approach to the data set J60. The results obtained which are summarized in the column "BBK" of Table 7.14 can be downloaded from the web at <http://scarlett.mathematik.uni-osnabrueck.de/research/or/rcpssp/rcpssp.html>. Note that the results refer to all instances of the data set, i.e., also the instances with RS = 1.0 are considered. In order to compute the deviations from optimality, the values yielded by SCATTER within one hour of computation time are used (cf. Section 7.5.2.2). The experiments have been performed on a SUN Sparc 20/801 workstation with a clockpulse of 80 MHz and 64 MB computer storage. The operating system is Solaris 3.5. For solving the LP-relaxations, the general purpose solver CPLEX 4.0 has been used. Unfortunately, no computation times are given.

Table 7.14. Destructive improvement vs. set covering based approaches (J60)

data set		BBK	BK	LB*	RPC _{ma}	RPC _{all}	RSPC _{all}
J60	av.dev	3.59	2.60	3.96	3.57	3.29	3.22
	max.dev	22.50	17.59	27.97	27.35	25.00	25.00
	av.cpu	—	5.00	0.32	0.38	0.79	8.20

The results reveal that also for J60, RPC_{ma} yields a bound quality comparable to the one obtained by the set covering based approach. If all bound arguments are employed, destructive improvement comes out first.

Brucker and Knust (1998) have combined the set covering based approach with destructive improvement. A restricted problem is reduced by applying techniques which have been introduced in Brucker et al. (1998) and which are similar to reduction by precedence. Subsequently, the maximal compatible sets are computed also considering time window information in order to decide whether certain jobs can be processed in parallel or not. Furthermore, the formulation (4.19)-(4.21), pp. 135, is reinforced exploiting this information. In order to keep the computational effort for examining trial lower bound values low, a binary search is applied (Remark 4.7, p. 140). An initial lower bound value is obtained by starting with LBC1. An upper bound value is computed by a priority-rule based heuristic.

The results obtained by this approach are contained in the column "BK" of Table 7.14. The computations have been performed on a SUN Ultra 2 workstation with 167 MHz clockpulse and 320 MB main storage. The operating system is Solaris 2.5. Table 7.14 shows that combining destructive improvement with the set covering based approach clearly performs best. However, the computational effort required for computing this bound argument is rather high though only a rough comparison is possible due to different computer systems used.

To summarize, all results presented above show that destructive improvement is a very successful method for computing high quality bounds.

7.4 Heuristic Procedures

In the sequel, we investigate the computational effectiveness of heuristic procedures for solving RCPSP and GRCPSP. In Section 7.4.1, the effectiveness of priority-rule based heuristics is reviewed whereas Section 7.4.2 is devoted to the examination of improvement procedures.

7.4.1 Priority-Rule Based Heuristics

First of all, an in-depth evaluation of priority-rule based heuristics for RCPSP is performed in the Sections 7.4.1.1 to 7.4.1.4 (cf. Section 5.2). In the first part, we restrict to giving the results yielded for SMFF for which optimal solutions are known to all instances. Though this is also true for PATT, the SMFF instances have been generated systematically and therefore allow for a more sound interpretation of the results. For J60, the computational experiments re-

vealed that the results do not provide much additional insight and, therefore, can be omitted. Finally, in Section 7.4.1.5 the performance of the priority rules being most successful for RCPSP is evaluated for GRCPSP.

7.4.1.1 Combinations of Scheduling Schemes and Priority Rules

In a first step, we compare the effectiveness of the scheduling schemes in combination with the different priority rules. The objective function value obtained for an instance by each combination represents an upper bound UB on the optimal project duration. The results of the experiment are compared with respect to the following measures:

- av.dev: average relative deviation of UB from optimality in %
- max.dev: maximum relative deviation of UB from optimality in %
- #opt: number of instances for which a combination finds an optimal solution
- #best: number of instances for which a priority rule finds the best solution among all rules using the same scheduling scheme and planning direction (multiple hits possible)

Relative deviations are computed by $(UB - CT)/CT$ with CT denoting the optimal project completion time of an instance. A summary of the results is presented in the Tables 7.15 and 7.16 which are subdivided according to Table 5.2, p. 182. The best results which have been yielded for each measure, scheduling scheme and group of rules are typeset in a bold manner. Furthermore, the shaded cells indicate for each combination of a scheduling scheme and a priority rule in which planning direction the best results are obtained. Note that we do not give computation times, because they are negligibly small.

Comparison of Scheduling Schemes. For the measures av.dev and max.dev, PSS obtains better results than SSS considering the same priority rule and planning direction except for a few exceptions (cf Table 7.15). This observation corresponds to those made, e.g., by Alvarez-Valdés and Tamarit (1989 b) and Kolisch (1996 b). However, analyzing the number of optimal instances found, SSS outperforms PSS for some priority rules when planning is done unidirectionally, i.e., either in forward or in backward direction (cf. Table 7.16). This is no longer true, if both scheduling schemes are applied bidirectionally, because PSS clearly comes out first. Taking into account the average and maximum de-

viations from optimality indicates that SSS yields solutions which are either very good or rather poor, whereas PSS tends to generate solutions with a smaller range of variations in the solution quality.

Table 7.15. Average and maximum deviations for SMFF

rule	av.dev						max.dev					
	SSS			PSS			SSS			PSS		
	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.
SPT	22.80	25.49	15.85	13.09	14.39	8.66	57.14	65.67	49.23	44.28	43.64	36.49
MIS	12.79	19.64	11.59	9.14	13.77	7.77	48.39	60.47	41.86	41.86	54.17	33.33
MTS	8.74	8.37	7.63	6.66	7.39	5.22	32.31	35.41	37.20	25.00	34.72	24.49
RPW	13.75	15.21	11.31	10.47	12.18	8.59	51.85	56.81	51.61	41.86	48.61	34.21
RPW*	7.04	7.20	7.14	6.20	6.71	4.78	30.91	32.35	37.20	32.81	30.19	23.68
EST	11.99	14.94	8.67	10.49	12.77	7.51	44.12	45.46	29.23	41.86	35.36	33.33
EFT	15.44	17.25	9.82	11.23	12.49	7.41	46.15	56.45	31.03	38.04	35.90	29.03
ESTD	11.04	14.20	8.25	9.34	17.53	8.10	36.77	46.05	27.38	41.86	55.10	39.54
EFTD	14.62	15.86	9.60	13.09	14.39	8.49	43.42	46.77	36.74	44.28	43.64	37.10
LST	6.56	6.73	6.84	6.04	6.94	5.25	33.33	33.33	30.43	32.81	27.69	26.56
LFT	7.44	7.33	6.69	5.86	6.78	4.49	33.90	33.72	37.21	24.14	31.71	21.73
MSLK	12.66	13.35	11.49	8.96	9.61	6.70	49.21	56.10	54.23	44.21	55.55	37.93
MSLD	13.27	13.78	11.57	6.04	6.94	5.34	50.85	59.32	45.24	32.81	27.69	26.56
GRD	15.85	16.73	15.85	12.33	14.19	9.48	59.52	51.16	59.52	44.07	55.10	38.46
WRUP	15.64	15.47	11.51	9.78	11.03	7.65	51.52	52.31	48.84	38.78	39.13	33.33
ACTRES	9.00	9.17	8.06	6.92	7.47	5.55	44.74	45.00	37.50	35.90	39.66	30.77
ACROS	9.80	9.81	8.27	6.98	7.44	5.12	42.19	37.29	37.50	38.10	31.75	38.10
TIMRES	7.38	7.20	7.65	6.32	6.95	5.26	33.33	30.88	35.89	32.81	34.70	23.53
TIMROS	7.19	6.69	6.92	6.09	6.39	4.86	43.08	28.57	38.46	32.81	28.57	23.44
IRSM	—	—	—	6.56	7.74	5.23	—	—	—	33.87	35.05	28.81
WCS	—	—	—	5.31	6.40	4.68	—	—	—	27.78	23.68	25.00
ACS	—	—	—	5.51	6.70	4.69	—	—	—	31.25	23.68	25.00
WCLS	6.42	6.95	7.96	—	—	—	32.26	32.35	32.14	—	—	—
RAND	19.79	21.58	17.62	13.09	15.02	8.75	73.13	61.29	52.94	46.15	50.00	44.07

Most surprisingly, employing different planning directions leads to contrary effects for both scheduling schemes. For SSS, no planning direction can be identified performing best. For the majority of priority rules, bidirectional planning comes out first. However, the most promising priority rules, such as RPW*, LST and WCLS, obtain their best results in case of unidirectional planning.

This indicates that SSS should always be applied in different planning directions.

For PSS, bidirectional planning clearly outperforms the unidirectional versions with respect to all measures. Considering the measure av.dev, it always yields smaller values than for unidirectional planning. Furthermore, it manages to find more optimal solutions irrespective of the priority rule used. This finding is strengthened considering the results obtained by backward planning. In the worst case, if restricting to an unidirectional planning direction as this is traditionally done, the project network can always be present in the "wrong" direction. Note that bidirectional planning yields nearly identical results when applied to the original or to the reversed project network. That is, it is less data dependent and, hence, more robust than the unidirectional versions. This is also underlined by the performance of the RAND rule where bidirectional planning always comes out first. Finally, it is worth mentioning that bidirectional PSS succeeds in finding a considerably larger number of optimal solutions than its unidirectional counterparts. For a possible explanation, we refer to Remark 5.10, p. 180.

Comparison of Priority Rules. For SSS, the priority rules RPW*, LST, TIMROS, and WCLS perform best which is true concerning all measures (cf. Tables 7.15 and 7.16). In forward direction, the rule WCLS comes out first. It yields the shortest project completion times for nearly half of the 360 instances and finds optimal solutions to more than a third of the instances (cf. the columns "best" and "#opt" in Table 7.16). Among the traditional rules, LST is identified as the best one thus confirming former research. In general, rules based on computing longest paths to the dummy end job seem to be most promising. This is partially due to the structural deficits of SSS outlined in Section 5.2.1.3, pp. 173. A further interesting observation is that for bidirectional planning a slightly different ranking of priority rules is obtained. For example, the LFT rule performs much better in this case and, thus, enters the set of promising rules.

For PSS, the same set of rules can be recommended complemented by the LFT rule. In contrast to SSS, the ranking of priority rules does not change considerably depending on the planning direction. For bidirectional planning, the WCS rule does not yield better results than the LFT rule and the RPW* rule. This observation is important because it shows that static rules like RPW* are able to cope with dynamic ones when planning bidirectionally. With an increasing

Table 7.16. Numbers of optimal and best solutions for SMFF

rule	#opt						#best					
	SSS			PSS			SSS			PSS		
	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.
SPT	17	16	40	32	36	71	21	17	45	67	58	106
MIS	52	35	60	57	46	83	61	37	66	116	79	121
MTS	93	96	100	70	73	106	108	122	112	138	134	163
RPW	62	51	72	59	57	81	73	59	80	110	98	113
RPW*	114	110	108	81	77	110	151	152	135	164	152	171
EST	29	29	64	35	37	75	40	39	83	75	61	110
EFT	15	21	46	30	39	68	24	25	64	69	64	104
ESTD	36	37	66	46	32	72	55	44	91	94	52	109
EFTD	17	27	48	32	36	73	25	33	65	67	58	106
LST	126	122	117	93	80	117	163	158	148	189	166	179
LFT	120	106	120	86	82	133	154	137	148	173	163	197
MSLK	78	73	79	84	68	104	91	84	85	148	124	143
MSLKD	77	73	75	93	80	115	89	80	80	189	166	174
GRD	43	41	43	39	46	69	54	46	49	74	75	96
WRUP	41	36	59	49	51	71	49	47	74	103	102	101
ACTRES	95	95	104	81	82	113	124	121	123	165	151	173
ACROS	81	85	91	67	73	110	104	110	116	146	134	168
TIMRES	112	105	108	85	82	129	148	144	132	174	163	175
TIMROS	117	115	122	86	87	125	144	161	151	170	170	190
IRSM	—	—	—	91	76	125	—	—	—	174	143	170
WCS	—	—	—	97	85	132	—	—	—	202	174	199
ACS	—	—	—	92	85	124	—	—	—	193	165	185
WCLS	133	124	106	—	—	—	171	154	122	—	—	—
RAND	23	29	33	35	36	65	28	30	35	68	64	93

number of jobs, updating priority values each time having scheduled a job may constitute a considerable computational effort. In our computational tests, the average computation times for applying the WCS rule showed to be approximately three times higher than when using the RPW* rule.

7.4.1.2 Influence of the Problem Structure

Comprehensive computational studies can be found in the literature comparing the effectiveness of priority-rule based heuristic procedures for solving RCPSP

(cf., e.g., Davis and Patterson (1975), Alvarez-Valdés and Tamarit (1989 b), and Kolisch (1996 b)). All these experiments reveal that for some instances solutions close to the optimum are determined whereas for others no priority-rule based heuristic yields satisfying results. Furthermore, depending on the structure of a problem instance to be solved, different combinations of scheduling schemes and priority rules perform best. For these reasons, problem instances are classified using complexity measures as introduced in Section 7.2.1 to allow examining the influence of the problem structure on the performance of priority-rule based heuristics more thoroughly.

According to the lower bound arguments, the resource strength has turned out to be the most significant measure for judging the complexity of instances in terms of the expected solution quality which can be yielded by priority-rule based procedures (cf. Kolisch (1996 b)). Since only SMFF is considered, which has been generated systematically as described in Section 7.2.2, the instances can be subdivided into three classes with a resource strength RS of 0.2, 0.5, and 0.7, respectively. Note that the 120 instances contained in each of these classes correspond to the ones of the classes C1, C2 and C3 considered in Section 7.3.3. In general, the largest deviations from optimality can be observed for RS = 0.2, the smallest ones for RS = 0.7 irrespective of the scheduling scheme and priority rule used.

The following Tables 7.17 to 7.19 show the results obtained for each class concerning the measures av.dev and max.dev. Only a selection of all rules providing the most interesting results is considered.

Table 7.17. Average and maximum deviations for RS = 0.2

rule	av.dev						max.dev					
	SSS			PSS			SSS			PSS		
	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.
RPW*	12.21	13.05	12.36	10.28	9.76	8.03	30.91	32.35	37.20	32.81	30.19	23.68
LST	12.34	12.96	13.26	10.80	10.88	9.58	33.33	33.33	30.43	32.81	27.69	25.00
LFT	13.45	13.33	12.71	9.31	10.15	8.03	33.90	33.72	37.20	24.14	28.81	21.27
WRUP	20.59	19.55	16.61	12.84	14.36	10.62	51.51	52.31	46.66	36.91	38.46	27.63
TIMRES	13.17	12.90	13.57	11.01	10.56	8.92	33.33	30.88	33.33	32.81	28.57	23.53
TIMROS	12.86	12.16	12.60	10.32	9.56	8.43	43.08	28.57	38.46	32.81	25.42	22.44
WCS	—	—	—	9.18	9.75	8.65	—	—	—	27.78	23.68	24.49
WCLS	11.90	12.93	12.96	—	—	—	32.26	32.35	32.14	—	—	—

For $RS = 0.2$ and SSS, the new WCLS rule yields the best results among all priority rules when applied in forward direction (cf. Table 7.17). Considering PSS, bidirectional planning is most successful for each priority rule. Using the RPW* or the LFT rule, the average deviation works out at only two thirds of the best result obtained by SSS and the WCLS rule. The WCS rule justifies its rather large computational effort only in forward direction.

Table 7.18. Average and maximum deviation for $RS = 0.5$

rule	av.dev						max.dev					
	SSS			PSS			SSS			PSS		
	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.
RPW*	5.87	6.07	6.09	4.93	6.14	3.80	24.44	21.57	26.66	18.37	23.81	11.90
LST	5.21	5.08	5.15	4.47	5.91	3.95	18.64	18.64	22.58	15.69	19.57	16.94
LFT	6.02	5.76	5.33	4.83	6.21	3.20	20.76	21.82	20.00	21.95	31.71	12.25
WRUP	15.25	15.10	11.25	9.84	10.72	7.47	48.84	37.21	48.84	38.78	36.21	33.33
TIMRES	6.18	5.79	6.27	4.90	5.85	4.35	22.22	20.51	35.89	28.89	17.07	18.42
TIMROS	6.11	5.28	6.09	5.08	5.70	4.06	22.22	20.51	22.22	21.95	21.43	16.94
WCS	—	—	—	3.98	5.66	3.30	—	—	—	15.39	18.61	16.94
WCLS	5.05	5.68	7.15	—	—	—	20.51	19.35	25.00	—	—	—

Considering $RS = 0.5$, PSS keeps on clearly outperforming SSS, in particular when it is applied bidirectionally (cf. Table 7.18). For this group of instances, the effect of different planning directions becomes apparent most clearly. The most striking differences can be observed for PSS in combination with the LFT rule. For the reversed precedence network (backward planning), the average deviation is nearly twice as much as with bidirectional planning. Comparing the maximum deviations, the lead of bidirectional planning becomes even more clear.

Increasing the resource strength to $RS = 0.7$ leads to a slightly different picture (cf. Table 7.19). Now, SSS can compete with PSS for most of the rules. This result is confirmed by the findings provided in Kolisch (1996 b).

Comparing the planning directions performing best for different resource strengths is interesting, too. For SSS, it nearly always changes except for the LFT rule and the WRUP rule whereas for PSS unidirectional planning never excels bidirectional planning. To resume, if only a single scheduling scheme is to be applied, PSS with bidirectional planning should be preferred.

Table 7.19. Average and maximum deviations for RS = 0.7

rule	av.dev						max.dev					
	SSS			PSS			SSS			PSS		
	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.	fwd.	bwd.	bid.
RPW*	2.91	2.32	2.82	3.33	4.14	2.44	17.19	20.31	20.31	16.33	23.68	13.63
LST	2.01	2.04	1.99	2.78	3.96	2.15	15.91	18.42	14.81	13.21	23.68	13.63
LFT	2.72	2.80	1.90	3.37	3.87	2.22	18.42	14.29	14.81	15.39	25.00	14.28
WRUP	10.88	11.62	11.25	6.52	7.89	4.74	47.62	45.24	23.81	28.07	39.13	22.81
TIMRES	2.65	2.78	2.98	2.95	4.40	2.44	22.64	18.42	26.41	11.36	28.57	13.63
TIMROS	2.45	2.49	1.89	2.77	3.82	1.99	20.41	18.42	12.76	14.29	28.57	13.63
WCS	—	—	—	2.71	3.68	2.03	—	—	—	12.25	23.68	13.63
WCLS	2.20	2.09	3.61	—	—	—	17.31	16.32	26.41	—	—	—

7.4.1.3 Multi-Pass Performance

In the following, the performance of the different scheduling schemes and planning directions when incorporated into a multi-pass priority-rule method (cf. Section 5.2.4, pp. 187) is examined. For this purpose, each scheduling scheme is applied in each planning direction combined with the subset of rules also considered for examining the influence of the problem structure (cf., e.g., Table 7.17). Among the solutions determined, the best one is chosen. Subsequently, we evaluate the benefits of employing a scheduling scheme within multi-planning direction methods (forward, backward, and bidirectional). Finally, the power of a multi-priority rule, multi-planning direction method called MPP combining the selected priority rules with both scheduling schemes and all planning directions is reviewed. Again no computation times are recorded because they are negligibly small. For example, performing MPP required a maximum computation time of 0.2 seconds per instance.

The results obtained are given in Table 7.20. In addition to SMFF, the data set PATT has been considered. First of all, the performance of both scheduling schemes can considerably be increased by planning in all "directions" (cf. the column "all" for both schemes) with respect to all measures. For PATT and SSS, this leads to a decrease of the average deviation by up to 38 % in comparison to only scheduling bidirectionally whereas for PATT and PSS a reduction of at most 45 % is achieved. Also for the data set SMFF, remarkable improvements are yielded. Comparing both scheduling schemes, PSS clearly outperforms SSS in the multiple direction version. However, the performance for

Table 7.20. Multi pass performance for the rules of Table 7.17

data set		SSS				PSS				MPP
		fwd.	bwd.	bid.	all	fwd.	bwd.	bid.	all	
PATT	av.dev	2.96	3.14	2.51	1.57	2.17	2.75	1.74	0.95	0.90
	max.dev	15.00	15.78	11.32	9.43	15.79	15.79	11.32	7.55	7.55
	#opt	42	47	47	65	56	51	62	78	79
SMFF	av.dev	4.08	4.17	3.87	2.58	3.77	4.40	2.75	1.81	1.45
	max.dev	22.61	27.94	21.54	17.24	17.77	22.81	17.10	14.43	10.44
	#opt	163	160	169	196	113	112	163	198	224

SMFF can be further increased by utilizing MPP, i.e., combining multi-planning direction SSS and multi-planning direction PSS. This indicates that both scheduling schemes have to be applied in order to obtain the best results possible.

In Section 5.2.4, adaptive sampling methods have been introduced which are considered to be the most effective priority-rule based approaches for solving RCPSP. In the sequel, we compare the performance of the adaptive search method (ASP) proposed by Kolisch and Drexl (1996) and the class based control scheme method (CCS) introduced by Schirmer and Riesenber (1997 a) to MPP. Due to different combinations of priority rules, scheduling schemes, and planning directions employed, MPP evaluates a total of 42 schedules. For ASP and CCS, computational results with sample sizes of 100, 1000, and 5000 schedules are available, respectively. In order to allow for an almost fair comparison, the results obtained for a sample size of 100 are presented, though this already corresponds to more than twice the number of schedules determined by MPP.

Table 7.21 contains results for three different data sets. The data set SMFF308 has been assembled from SMFF (cf., e.g., Kolisch (1995, section 5.4.1)) containing those instances for which optimal solutions have been known at the time of developing ASP.

Table 7.21 shows that for the data set PATT, ASP yields better results than MPP concerning the measures av.dev and #opt (no results are available for CCS). By the way of contrast, MPP clearly outperforms ASP for the more challenging SMFF308 data set with respect to all measures. Even applying the more evolved procedure CCS, adaptive random sampling is not able to cope with MPP as underlined by the results for SMFF308 and SMFF. This leads to the

Table 7.21. Comparison to adaptive sampling procedures

data set		ASP	CCS	MPP
PATT	av.dev	0.78	—	0.90
	max.dev	9.43	—	7.55
	#opt	84	—	79
SMFF308	av.dev	1.22	1.05	0.91
	max.dev	11.84	11.11	8.33
	#opt	204	209	222
SMFF	av.dev	—	1.53	1.45
	max.dev	—	11.11	10.44
	#opt	—	213	224

conclusion that if only a small number of schedules can be evaluated due to, e.g., the available computation time being restricted, a multi-priority rule, multi-planning direction method should be given preference to an adaptive sampling approach. However, it is important to state, that the performance of the latter approaches can considerably be increased by allowing for a larger sample size. Furthermore, it has not been evaluated yet how these approaches can benefit from incorporating different planning directions.

7.4.1.4 Comparison to Proprietary Heuristics of Standard Software

In the sequel, we give results yielded by the (unknown) proprietary heuristics contained in Computer Associates (CA) Superproject 4.0a and Microsoft Project 4.0, two of the most popular standard software packages for project management. Both packages include programming languages (CA Realizer and Microsoft Visual Basic for Applications) which allow for integrating the priority-rule based procedures presented in Section 5.2 without considerable effort. Hence, for project managers and software developers it is interesting whether this effort pays off.

The results obtained for the data sets PATT and SMFF are given in Table 7.22. For sake of convenience, we also partially repeat the results of Table 7.20.

Considering the two software packages, the heuristic of CA Superproject 4.0a clearly outperforms the one of MS Project 4.0. Nevertheless, the results yielded by both packages should alert any project manager using the proprietary heuristics. Taking into account the rather small size of the projects, a maximum deviation of over 50% as in case of MS Project 4.0 can not be tolerated. In particu-

Table 7.22. Comparison to proprietary heuristics of standard software

data set		SSS		PSS		MPP	software	
		bid.	all	bid.	all		CA	MS
PATT	av.dev	2.51	1.57	1.74	0.95	0.90	4.60	6.43
	max.dev	11.32	9.43	11.32	7.55	7.55	20.75	35.00
	#opt	47	65	62	78	79	32	29
SMFF	av.dev	3.87	2.58	2.75	1.81	1.45	7.28	8.87
	max.dev	21.54	17.24	17.10	14.43	10.44	36.23	50.75
	#opt	169	196	163	198	224	106	90

lar, this is true considering the fact that the most complex version of our multi-pass priority-rule based heuristic MPP required a maximum computation time of 0.2 seconds.

In order to keep the effort required for integrating the heuristics into the software packages small, one may only want to implement a single scheduling scheme. In this case, the parallel scheduling scheme PSS should be preferred.

7.4.1.5 Results for GRCPSP

In the open literature, no computational experiments evaluating the effectiveness of priority-rule based heuristics for solving GRCPSP have been reported so far. In the following, we will close this gap by applying the priority-rule based heuristics introduced in Section 5.2 to the GRCPSP data sets defined in Section 7.2.3. In order to summarize the results, we use the same measures as before. Accordingly, the measure "#best" refers to the best solutions obtained by a single scheduling scheme in combination with the selected subset of priority rules.

Results are only given for those priority rules which have been identified to be most promising for RCPSP (cf. Table 7.17). Among those rules, WRUP, TIMRES, and TIMROS have been excluded because they assume constant resource availabilities (see p. 187). Computational experiments show that all excluded rules introduced in Section 5.2.3 perform considerably worse than the chosen ones. Furthermore, for GRCPSP, the scheduling schemes can only be applied within forward planning. Before giving detailed results, it is important to state that GRCPSP represents an optimization problem being NP-hard in the strong sense, such that heuristic procedures might fail to determine a feasible

solution at all, in particular when restrictive release and due dates have to be considered. Therefore, results for the data set P1/R3/T1 are omitted.

Serial Scheduling Scheme. The Tables 7.23 and 7.24 contain the results yielded by SSS concerning the measures already evaluated for RCPSP. The best results obtained for each measure are typeset bold.

Table 7.23. Average and maximum deviations for SSS

data set	av.dev				max.dev			
	RPW*	LST	LFT	WCLS	RPW*	LST	LFT	WCLS
P0/R0/T0	63.32	52.28	66.53	40.12	153.49	134.04	150.00	132.76
P0/R1/T0	64.10	52.69	66.84	39.79	153.49	136.84	150.00	132.76
P0/R2/T0	58.91	49.36	59.13	37.68	145.45	134.04	142.39	132.08
P0/R3/T0	35.89	32.43	34.39	28.71	145.45	110.84	127.27	110.84
P0/R4/T0	7.04	6.56	7.44	6.42	30.91	33.33	33.90	32.26
P0/RSP/T0	14.44	14.50	15.06	13.80	40.74	37.86	40.28	37.86
P1/R3/T0	37.57	33.01	34.55	28.79	169.09	110.84	127.27	110.84
P1/R4/T0	8.41	7.13	8.35	6.64	33.33	38.46	38.98	38.46
P1/RSP/T0	14.17	14.39	14.94	13.65	54.26	39.82	44.05	33.72

First of all, it is apparent that the average and maximum deviations which have to be accepted for the data sets with resource profiles of the types R0 to R3 are extraordinary large. For the WCLS rule performing best, still a maximum deviation of 132.76 % occurs. These large deviations may be explained as follows. Within the data sets with resource profiles of the types R0 to R3, a job j can possibly be processed only in a few period intervals of the planning horizon due to the resource availability in the other periods not being sufficient (cf. Remark 7.1, p. 273). That is, if a partial schedule is determined by the heuristic such that job j can not be started early enough to be processed in a certain interval, job j has to be postponed for a considerably number of periods until the next suitable interval begins. As a consequence, also its successors have to be delayed such that the same effect may again occur for those jobs and, hence, may be multiplied. The observation is underlined by considering the number of instances solved to optimality (cf. Table 7.25). This number being rather large for these data sets (containing 360 instances each) indicates that the procedures either find a solution close to the optimum or, otherwise, miss it by far.

By the way of contrast, the deviations from optimality are considerably smaller for the data sets based on resource profiles of the type R4 and RSP where each

Table 7.24. Number of optimal and best solutions for SSS

data set	#opt				#best			
	RPW*	LST	LFT	WCLS	RPW*	LST	LFT	WCLS
P0/R0/T0	105	138	93	186	167	196	141	266
P0/R1/T0	98	133	88	182	165	202	141	272
P0/R2/T0	82	113	75	159	153	195	138	276
P0/R3/T0	83	102	86	117	177	213	171	247
P0/R4/T0	114	126	120	133	207	222	195	245
P0/RSP/T0	26	25	34	27	155	140	141	169
P1/R3/T0	76	98	82	115	174	209	175	245
P1/R4/T0	98	122	114	125	174	227	181	244
P1/RSP/T0	19	19	36	22	167	136	130	169

job can basically be processed in any period of the planning horizon (cf. Remark 7.1, p. 273). The data sets with a constant resource availability (RCPSP) represent the most simple ones. For the data sets with resource profiles RSP, the numbers of instances which are solved to optimality is very small. Hence, for these data sets the contradictory effect of the one described above for the data sets with resource profiles R0 to R3 can be observed. In contrast to the different types of resource profiles, the type of project network defined has no considerable influence on the effectiveness of the solution procedures. This can, e.g., be verified by comparing the results obtained for the data sets P0/R3/T0 and P1/R3/T0.

Among the priority rules, the new regret based rule WCLS introduced by Klein (1998) outperforms all other rules concerning the measure av.dev and #best. Also concerning the remaining measures it comes out first in almost all cases.

Parallel Scheduling Scheme. Considering the average deviations, PSS clearly yields better results than SSS for all data sets whereas both scheduling schemes perform similarly concerning the maximum deviations (cf. Table 7.25). The number of instances solved to optimality is much larger for PSS than for SSS for the data sets with resource profiles R0 to R3 (cf. Table 7.26). However, for the remaining data sets based on resource profiles R4 and RSP preference should be given to SSS with respect to this measure. In general, the relationships between data sets and heuristic performance observed for SSS also can be noticed for PSS.

Table 7.25. Average and maximum deviations for PSS

data set	av.dev				max.dev			
	RPW*	LST	LFT	WCS	RPW*	LST	LFT	WCS
P0/R0/T0	34.89	26.95	36.07	18.03	144.57	133.90	154.55	130.16
P0/R1/T0	36.75	28.62	37.69	18.58	135.00	135.71	137.37	130.16
P0/R2/T0	37.16	30.53	37.30	24.02	140.22	140.22	145.71	131.34
P0/R3/T0	28.42	25.67	27.12	23.11	134.55	113.58	138.38	113.58
P0/R4/T0	6.20	6.04	5.86	5.31	32.81	32.81	24.14	27.28
P0/RSP/T0	11.91	12.36	11.77	11.46	32.53	36.51	33.33	30.21
P1/R3/T0	29.11	26.38	27.42	23.63	133.70	120.99	145.71	120.99
P1/R4/T0	6.98	6.29	6.02	5.55	34.48	34.48	23.53	26.47
P1/RSP/T0	11.60	11.70	11.31	10.93	30.33	32.31	30.12	30.12

Examining the different priority rules, the regret based WCS rule introduced by Kolisch (1996 a) excels all other rules. This is in particular true for the data sets with resource profiles R0 to R3.

Table 7.26. Number of optimal and best solutions for PSS

data set	#opt				#best			
	RPW*	LST	LFT	WCS	RPW*	LST	LFT	WCS
P0/R0/T0	213	243	204	277	239	270	230	311
P0/R1/T0	206	238	198	274	233	266	226	312
P0/R2/T0	181	215	173	235	228	268	221	303
P0/R3/T0	122	155	128	158	217	250	222	276
P0/R4/T0	81	93	86	97	227	240	229	272
P0/RSP/T0	15	16	23	21	178	163	183	200
P1/R3/T0	120	153	137	161	214	248	223	268
P1/R4/T0	82	97	95	108	217	240	227	274
P1/RSP/T0	19	14	28	25	174	159	181	201

Multi Pass Performance. Finally, we examine the effectiveness of combining different scheduling schemes and priority rules into a multi-pass priority-rule heuristic MPP. For this purpose, the best solutions yielded by any of the priority rules employed in each scheduling scheme are evaluated, respectively (cf. Tables 7.23 and 7.25). Furthermore, the overall best solution obtained by any of the 8 priority-rule based heuristics is considered, i.e., combining SSS and PSS. The results yielded are summarized in Table 7.27. They reveal that for each scheduling scheme the different priority rules used complement each other

well. The average deviations are considerably reduced and the number of instances solved to optimality is increased. However, the maximum deviations which can be observed for the data sets with resource profiles R0 to R3 are still unacceptably large. This is still true, if both scheduling schemes are applied. In general, PSS clearly outperforms SSS except for P0/R4/T0 and P1/R4/T0.

Table 7.27. Multi pass performance for GRCPSP

data set	SSS			PSS			MPP		
	av.dev	max.dev	#opt	av.dev	max.dev	#opt	av.dev	max.dev	#opt
P0/R0/T0	25.93	123.44	232	10.77	106.52	302	9.41	106.52	307
P0/R1/T0	27.51	123.44	223	11.94	112.50	297	10.25	100.00	302
P0/R2/T0	29.14	115.15	190	17.70	131.34	262	12.88	100.00	278
P0/R3/T0	22.95	110.84	147	18.02	113.58	186	12.40	103.61	224
P0/R4/T0	4.44	23.81	157	4.18	18.28	107	2.74	16.30	178
P0/RSP/T0	13.80	28.81	47	9.04	28.79	28	7.54	23.08	56
P1/R3/T0	23.82	110.84	146	18.37	120.99	189	13.32	103.61	225
P1/R4/T0	4.85	23.53	155	4.41	21.13	116	2.98	19.12	182
P1/RSP/T0	10.20	28.92	47	8.54	23.16	30	7.31	22.95	56

7.4.2 The Tabu Search Procedure RETAPS

In the sequel, we examine the effectiveness of the tabu search procedure RETAPS, which is introduced in Section 5.3.2. For this purpose RETAPS is configured as follows. The candidate list strategy which randomly determines and evaluates a subset of all possible swap moves is applied (cf. Section 5.3.2.1, pp. 198). Tabu management and diversification are performed as described in Section 5.3.2.2. That is, a diversifying move which randomly generates a new start solution is executed each time three solutions have been visited twice. The initial feasible solution is constructed using SSS and the LST rule (cf. Table 5.2, p. 182).

In a first step, detailed computational results are presented and analyzed for the different GRCPSP data sets introduced in Section 7.2.3. This also includes the data set P1/R3/T1 which considers prespecified release and due dates for jobs. Note that RETAPS manages to determine feasible solutions for all instances of this data set which is not possible with priority-rule based heuristics. Subsequently, comparisons to multi-pass priority-rule based heuristics as well as to other meta-heuristic based approaches for RCPSP are performed. Further com-

putational results are given in Section 7.5.2.3, where RETAPS is combined with the branch and bound procedure SCATTER.

7.4.2.1 Analysis of GRCPSP Performance

Besides the information also evaluated in the former sections, the following measures are additionally provided:

- av.cpu: average computation time required per instance (in seconds)
- av.dm: average number of diversifying moves performed
- av.am: average number of examined moves which are admissible in each iteration (determined after 3000 iterations)

In order to trace the development of the solution process, the performance data are recorded after 500, 1000, 2000, and 3000 iterations (except for av.am). In case a solution is found for which optimality can be proven due to the project completion time being equal to the length of the critical path LBC1', the procedure stops prematurely. The results obtained are given in the Tables 7.28 and 7.29.

Considering the average deviations from optimality, RETAPS performs very well. Already after 1000 iterations, an average deviation of at most 0.83 % is obtained for all data sets. For those data sets with resource profiles R0 to R2 being very hard to solve satisfactorily for priority-rule based heuristics, optimal solutions for all instances are already determined after 500 iterations (7 seconds of computation time on average). By the way of contrast, examining the measure max.dev shows that large maximum deviations have to be accepted for those data sets with resource profiles R3 even after 3000 iterations. The maximum deviations being exactly 100 % is only surprising at the first glance but can be explained by the generation of the resource profiles. For some of the instances concerned, certain jobs can only be scheduled in exactly one period interval within each of the time intervals $[0, CT]$, $[CT + 1, 2 \cdot CT]$, and so on with CT denoting the optimal project completion time of the underlying SMFF instance (cf. pp. 271). If the first interval is missed, the next interval must be used for this job thereby prolonging the schedule by CT (= 100 %).

However, taking into account the number of optimally solved instances indicates that this observation refers only to a few instances. Indeed, the data set P0/R3/T0 contains only one instance for which no satisfying solution is ob-

Table 7.28. Tabu search results for project networks P0

data set		R0/T0	R1/T0	R2/T0	R3/T0	R4/T0	RSP/T0
av.dev	500	0.00	0.00	0.00	0.99	0.08	0.79
	1000	0.00	0.00	0.00	0.80	0.04	0.57
	2000	0.00	0.00	0.00	0.67	0.02	0.42
	3000	0.00	0.00	0.00	0.67	0.01	0.35
max.dev	500	0.00	0.00	0.00	92.39	3.13	10.94
	1000	0.00	0.00	0.00	92.39	1.79	10.19
	2000	0.00	0.00	0.00	92.39	1.79	10.19
	3000	0.00	0.00	0.00	92.39	1.56	10.19
#opt	500	360	360	360	348	343	254
	1000	360	360	360	352	348	270
	2000	360	360	360	355	354	286
	3000	360	360	360	356	356	296
av.cpu	500	7.24	7.30	7.10	6.38	3.58	14.87
	1000	14.52	14.66	14.27	12.73	7.14	29.65
	2000	29.05	29.35	28.54	25.36	14.25	59.06
	3000	43.58	44.02	42.78	39.97	21.35	88.44
av.dm	500	4.21	3.91	3.45	2.77	2.37	1.61
	1000	7.74	7.11	6.16	4.96	4.32	2.91
	2000	14.83	13.58	11.65	9.32	9.27	5.56
	3000	21.89	20.06	17.12	13.63	12.23	8.14
av.am (3000 It.)		11.64	12.27	14.65	17.43	17.63	27.93

tained whereas this is true for two instances of the data sets P1/R3/T0 and P1/R3/T1, respectively. Since these instances can always be solved by the exact procedure SCATTER within very short time, applying a combined heuristic for these data sets may be promising (cf. Section 7.5.2.3). Allowing for up to 3000 iterations only pays off for the data sets with resource profiles RSP.

Concerning the average computation times (av.cpu) required for performing a certain number of iterations, it is interesting to state that they strongly depend on the data sets considered. For example, executing a certain number of iterations for the data sets with resource profiles of the type RSP requires four times as much computation time as for the data sets with resource profiles R4. This is due to the increased effort which arises when constructing a schedule considering resource profiles where a large number of fluctuations occur.

Table 7.29. Tabu search results for project networks P1

data set		R3/T0	R3/T1	R4/T0	RSP/T0
av.dev	500	0.77	1.29	0.12	0.59
	1000	0.57	0.83	0.07	0.41
	2000	0.51	0.72	0.03	0.27
	3000	0.47	0.72	0.02	0.20
max.dev	500	100.00	100.00	3.26	6.36
	1000	100.00	100.00	3.17	6.36
	2000	100.00	100.00	1.61	6.36
	3000	100.00	100.00	1.56	6.36
#opt	500	349	349	334	263
	1000	353	352	344	285
	2000	355	356	352	304
	3000	357	356	354	313
av.cpu	500	5.64	5.67	3.17	14.59
	1000	11.28	11.32	6.31	29.12
	2000	22.55	22.61	12.57	58.05
	3000	33.76	33.89	18.81	86.97
av.dm	500	2.41	2.36	2.26	1.80
	1000	4.36	4.24	4.19	3.15
	2000	8.20	7.96	8.03	5.83
	3000	12.03	11.70	11.86	8.54
av.am		15.11	15.12	15.38	27.39

Furthermore, a strong relationship between the data sets and the development of the search process can be observed. For the data sets with the rather restrictive resource profiles R0 and R1, a diversifying move is initiated approximately every 150 iterations, whereas for the data sets with resource profile RSP such a move is executed only every 350 iterations. This indicates that for the first type of instances the number of promising solutions is rather small such that cycling occurs early. This observation is underlined considering the average number of examined moves which are admissible in each iteration. Due to applying the candidate list strategy described in Section 5.3.2.1, the number of moves evaluated is approximately equal to $2 \cdot n$, if the previous move has not been deteriorating, or to n , otherwise. That is, for the data sets on hand at most 60 neighbors are considered in each iteration. For the data sets with resource profile RSP, about twice as much moves are admissible as for those with resource profiles R0 and R1.

7.4.2.2 Comparing RETAPS to Multi-Pass Heuristics

Since no other meta-heuristic based procedures are available for solving GRCPSP, we compare the performance of RETAPS to the one of the multi-pass heuristic MPP described in Section 7.4.1.5 using both, the serial and the parallel scheduling scheme. Note that the data set P1/R3/T1 is excluded. For the data set P0/R4/T0 being equal to SMFF, the multi-pass heuristic MPP discussed in Section 7.4.1.3 is applied (cf. Table 7.20, p. 294).

Table 7.30 contains results yielded by the multi-pass priority-rule based heuristics (cf. Tables 7.20 and 7.28) as well as by RETAPS after 1 and 5 seconds of computation time, respectively. Of course, such a comparison is not fair due to the multi-pass heuristic requiring much less computation time. However, it reveals how much gain in solution quality can be realized if an amount of computation time is provided which is also acceptable when using a project management software package. The experiments show that already after 1 second of computation time quite reasonable results are achieved by RETAPS at least considering the measure av.dev. The maximum deviations are still very large though they are caused by a very few instances only, as discussed earlier. In order to detect such instances, additionally some lower bound arguments may be applied. If the gap between the lower bound value obtained and the best project completion time determined is too large, some additional computation time can be provided.

Table 7.30. Multi-pass heuristics vs. RETAPS

data set	Multi-pass			RETAPS (1s)			RETAPS (5s)		
	av.dev	max.dev	#opt	av.dev	max.dev	#opt	av.dev	max.dev	#opt
P0/R0/T0	9.41	106.52	307	1.37	84.44	350	0.08	29.41	359
P0/R1/T0	10.25	100.00	302	1.35	100.00	351	0.00	0.00	360
P0/R2/T0	12.88	100.00	278	1.65	100.00	347	0.58	100.00	357
P0/R3/T0	12.40	103.61	224	3.82	92.39	285	1.53	92.39	337
P0/R4/T0	1.45	10.44	224	0.36	5.88	308	0.11	3.45	337
P0/RSP/T0	7.54	23.08	56	2.33	12.94	150	1.29	11.82	208
P1/R3/T0	13.32	103.61	225	3.58	100.00	317	1.71	100.00	334
P1/R4/T0	2.98	19.12	182	0.37	6.58	308	0.15	4.41	330
P1/RSP/T0	7.31	22.95	56	2.15	9.46	170	1.09	9.46	223

7.4.2.3 Comparing RETAPS to Other Heuristic Procedures for RCPSP

Finally, we examine the efficiency of RETAPS in comparison to adaptive sampling procedures and meta-heuristic based approaches developed for RCPSP (cf. Sections 5.2.4 and 5.3.3). For this purpose, we refer to computational experiments the results of which are presented by Hartmann and Kolisch (1998) and Kolisch and Hartmann (1999). Within their studies, most of the recent approaches are included (cf. Table 7.31). In order to allow for a fair comparison of the different procedures, the solution quality obtained by each procedure after evaluating 1000 and 5000 different schedules, respectively, has been determined for the data sets SMFF and J60. Note that in both cases also the 120 trivial instances with RS = 1 are included. Limiting the number of schedules to be evaluated rather than, e.g., the computation time has the advantage that the results yielded by a procedure do not depend on the computer on which it has been executed or on the efficiency of its implementation. However, the number of schedules allowed to be examined is rather small. Considering, e.g., the data set J60, RETAPS evaluates either 60 or 120 moves in each iteration due to the candidate list strategy employed (see pp. 202). That is, for 1000 schedules only between 8 and 16 iterations are performed.

For SMFF, the solution quality is measured by computing the average deviations from optimality. By way of contrast, for J60, not all instances have yet been solved to optimality. Therefore, the average deviations refer to the critical path bound LBC1. The results obtained are given in Table 7.31. In the second column, the type of meta-heuristic employed within each approach is denoted (adaptive sampling (AS), genetic algorithm (GA), simulated annealing (SA), and tabu search (TS)). First of all, the results reveal that RETAPS is one of the most efficient procedures for solving RCPSP on hand, though it has been designed for solving the more general GRCPSP. Among the remaining procedures, the ones of Bouleimen and Lecocq (1998) and Hartmann (1998) perform best (cf. Section 5.3.3, pp. 208). For SMFF and 1000 schedules, RETAPS comes out second behind the approach of Bouleimen and Lecocq (1998). However, for 5000 schedules it already yields better results than all other procedures.

For J60, RETAPS is slightly outperformed by the procedures of Bouleimen and Lecocq (1998) and Hartmann (1998) whereas it yields better results than all remaining procedures. In particular, smaller average deviations are obtained than by the tabu search procedure of Baar et al. (1998) (cf. Section 5.3.3). The better

Table 7.31. Comparison of heuristic procedures for RCPSP

source / procedure	Type	SMFF		J60	
		1000	5000	1000	5000
RETAPS	TS	0.42	0.17	12.77	12.03
Baar et al. (1998)	TS	0.86	0.44	13.80	13.48
Bouleimen and Lecocq (1998)	SA	0.38	0.23	12.75	11.90
Hartmann (1998)	GA	0.54	0.25	12.68	11.89
Kolisch and Drexl (1996)	AS	0.74	0.52	13.51	13.06
Leon and Balakrishnan (1995)	GA	2.08	1.29	14.33	13.49
Schirmer and Riesenbergs (1997 a)	AS	0.71	0.59	13.02	12.62
Schirmer (1998)	AS	0.65	0.44	12.94	12.59

performance of the procedures by Bouleimen and Lecocq (1998) and Hartmann (1998) may partially be explained as follows. These approaches are based on the meta-heuristics genetic algorithms and simulated annealing, respectively, which both contain a random device as basic component (cf. Section 5.3.3). Therefore, when the number of solutions to be evaluated is rather small, they effectively scan large parts of the solution space such that good solutions are obtained in short time. By the way of contrast, tabu search uses some type of memory in order to guide the search process. Hence, the solution space can be examined more systematically. However, this advantage does not pay off before a sufficiently large number of iterations has been considered. As stated above, at most 16 iterations are performed for J60.

7.5 Exact Procedures

In this section, we describe computational experiments performed to evaluate the effectiveness of our exact procedures for solving RCPSP and GRCPSP. The section is outlined as follows. First of all, the branch and bound procedure PROGRESS is examined in detail in Section 7.5.1. Subsequently, the benefits provided by integrating scattered branch and bound into this procedure are evaluated in Section 7.5.2.

7.5.1 The Branch and Bound Procedure PROGRESS

In the sequel, the performance of PROGRESS (cf. Section 6.2) is compared to the most efficient procedure GDH currently on hand for solving GRCPSP which

has been proposed by Demeulemeester and Herroelen (1997 a) (cf. Section 6.4.3, pp. 256). Since this comparison reveals PROGRESS to be the superior approach, its efficiency is subsequently analyzed in detail.

7.5.1.1 Comparing PROGRESS to GDH

In a first step, we compare the efficiency of the procedures PROGRESS and GDH by applying them to the data sets described in Section 7.2.3. For this purpose, the original code of the procedure GDH is employed which has been provided as an executable version by its developers. Note that it differs from the one described in Demeulemeester and Herroelen (1997 a) by already integrating the 32-bit features which have also been used in the new version of their RCPSP code (cf. Demeulemeester and Herroelen (1997 b), de Reyck et al. (1999), and Section 7.1). Since GDH is not able to solve all problem instances to optimality in reasonable time, a time limit of 300 cpu seconds is used (excluding input and output operations). After exceeding this limit, a *time out* occurs and the procedures stop before finding a proven optimum CT on which the best objective function value obtained so far represents an upper bound UB. For the experiment, the following version PRO-LLB of PROGRESS is used. The lower bound arguments proposed in Section 6.2.3 are omitted, because GDH does not use any of these bound arguments. By the way of contrast, all reduction and dominance rules of Section 6.2.4 are applied. The parameter ζ of the extended schedule storing rule is set to 2.

The results of the experiment are summarized in the Tables 7.32 and 7.33 which contain the following information:

- #opt: number of instances for which an optimal solution is found and proven
- av.dev: average relative deviation of UB from optimality in %
(relative deviation: $(UB - CT)/CT \cdot 100\%$)
- max.dev: maximum relative deviation of UB from optimality in %
- av.cpu: average computation time in seconds
- max.cpu: maximum computation time in seconds

The average computation times include the time-out cases with the lower bound time of 300 seconds.

Table 7.32. Comparing GDH to PRO-LLB for project networks P0

data set		R0/T0	R1/T0	R2/T0	R3/T0	R4/T0	RSP/T0
#opt	GDH	346	345	343	339	350	294
	PRO-LLB	360	360	360	360	358	360
av.dev	GDH	2.42	2.53	2.32	1.89	0.07	1.12
	PRO-LLB	0.00	0.00	0.00	0.00	0.00	0.00
max.dev	GDH	98.53	98.53	70.00	53.45	8.06	29.73
	PRO-LLB	0.00	0.00	0.00	0.00	0.00	0.00
av.cpu	GDH	14.61	15.38	18.06	21.79	10.19	66.63
	PRO-LLB	0.30	0.31	0.82	2.75	4.52	14.72
max.cpu	GDH	300.00	300.00	300.00	300.00	300.00	300.00
	PRO-LLB	42.19	42.76	60.28	148.97	300.00	272.83

For both procedures, the results show that the complexity of a data set essentially depends on the chosen type of resource profile rather than on the type of project network or the presence of time windows. They also indicate that solving instances tends to become easier if they contain minimum time lags which do not correspond to usual finish-to-start ones (project networks P1). Furthermore, considering time windows (T1) leads to a further decrease in the computational complexity of the respective instances. Both findings have already been observed in Demeulemeester and Herroelen (1997 a). The data sets with resource profile RSP clearly contain the most difficult instances.

Table 7.33. Comparing GDH to PRO-LLB for project networks P1

data set		R3/T0	R3/T1	R4/T0	RSP/T0
#opt	GDH	344	351	353	304
	PRO-LLB	360	360	359	360
av.dev	GDH	1.55	0.70	0.08	0.77
	PRO-LLB	0.00	0.00	0.00	0.00
max.dev	GDH	67.14	39.66	8.06	20.24
	PRO-LLB	0.00	0.00	0.00	0.00
av.cpu	GDH	18.68	10.41	7.21	61.66
	PRO-LLB	2.29	1.12	3.31	10.03
max.cpu	GDH	300.00	300.00	300.00	300.00
	PRO-LLB	123.02	82.90	300.00	156.40

Comparing the procedures, PRO-LLB outperforms GDH with respect to all measures. It determines optimal solutions to the instances of all data sets though it

is not able to prove the optimality of three of them. GDH fails to find proven optima for a total of 231 instances with a maximum relative deviation of 98.53% from the optimal project completion time CT. Also concerning the computation times, PRO-LLB comes out much better. For resource profiles of the type R0 and R1 it is more than 40 times faster than GDH. Note that, e.g., for the first data set P0/R0/T0 only one of the remaining 14 instances can additionally be solved to optimality within an increased time limit of 3600 seconds then resulting in an average computation time of 136.49 seconds. The fact that GDH obtains the smallest average computation times for the data set based on the original SMFF (R4) instances indicates that its rather poor performance in comparison to PRO-LLB is partly due to the restrictiveness of the GDH cutset rules (cf. Section 6.2.4.4, pp. 232). For the instances with resource profiles of type R4, no changes in the resource availability occur such that all partial schedules can be compared for dominance purposes by the cutset rule as proposed in Demeulemeester and Herroelen (1997 a) for RCPSP.

7.5.1.2 Analyzing the Efficiency of PROGRESS

In this section, the different components of PROGRESS are analyzed concerning their contribution to its good performance. First, we examine the effects of using the lower bound arguments presented in Section 6.2.3, pp. 226. Afterwards, different versions of the schedule storing rules are evaluated (cf. Section 6.2.4.4, pp. 232). Finally, the local lower bound method is compared to other depth-first search branching strategies such as DFSB and DFSL.

For these computational tests, we further reduce the number of data sets, because the less challenging ones with resource profiles R0 to R2 only confirm the results obtained for the remaining ones. The resource profile R3 is combined with P0 and P1 as well as with T0 and T1 in order to examine the influence of minimum time lags and time windows, respectively. The resource profile types R4 and RSP are only considered with project networks of type P0. All instances are solved to optimality, i.e., the computation time is not limited for any version of PROGRESS. The following measures are determined in addition to those already introduced:

- av.node: average number of nodes branched per instance
- av.stored: average number of partial schedules stored by the schedule storing rules (per instance)

The number of partial schedules which have to be stored by the schedule storing rules is important if the available computer memory is a critical resource.

Lower Bound Arguments. We compare three different versions of PROGRESS. One version is PRO-LLB which does not contain any bound arguments except for the implicit bound computation integrated in the local lower bound method (LBC1'). The other two versions PRO-LB22 and PRO-LB223 apply the capacity bound LBC2' and the one-machine bound LBM2' in each node in order to compute local lower bound values. PRO-LB223 additionally computes the two-machine bound LBM3' in each node.

Table 7.34. Efficiency of lower bound arguments

data set		P0/R3/T0	P1/R3/T0	P1/R3/T1	P0/R4/T0	P0/RSP/T0
av.cpu	PRO-LLB	2.75	2.29	1.12	4.77	14.72
	PRO-LB22	2.75	2.34	1.05	4.01	15.80
	PRO-LB223	3.99	3.53	1.49	5.59	22.17
max.cpu	PRO-LLB	148.97	123.02	82.90	363.69	272.83
	PRO-LB22	167.36	138.09	84.42	360.76	308.42
	PRO-LB223	238.60	197.84	120.15	541.50	421.78
av.nodes	PRO-LLB	4,712	3,965	1,887	8,356	20,192
	PRO-LB22	4,026	3,453	1,583	5,277	18,963
	PRO-LB223	3,991	3,404	1,564	4,255	18,907
av.stored	PRO-LLB	1,133	956	522	2,784	4,813
	PRO-LB22	964	855	446	1,936	4,623
	PRO-LB223	958	836	442	1,646	4,617

The results in Table 7.34 show that applying the lower bound arguments does not lead to a general reduction of computation times (the best result for each measure is typeset bold). By the way of contrast, for PRO-LB223, the average and maximum computation times are even larger than those of PRO-LLB for all data sets. However, computing the bound arguments always succeeds in considerably reducing the average number of nodes to be branched, in particular for the original SMFF (P0/R4/T0) data set which indicates that they are effective if the resource availabilities are almost constant. This is due to the fact that they consider the maximum availability of each resource type during the whole planning horizon for deciding whether a subset of jobs can be processed in parallel or not. Nevertheless, integrating the local lower bound arguments into

PROGRESS might be justified in order to keep the memory requirements of the schedule storing rules low.

Branching Strategies. In the following, we examine the different depth-first search strategies, discussed in Section 6.1.2, pp. 216, concerning their success in guiding the search process. The depth-first search version with complete branching PRO-DFSB yields the best results if the subproblems of a node are sorted according to the best local lower bound values obtained by applying the simple initial bound argument LBC1' as well as the capacity bound LBC2' and the one-machine bound LBM2'. Therefore, we compare PRO-DFSB to the laser search version PRO-DFSL also applying these bound arguments as well as to PRO-LB22. The dominance rules are all used as in the tests before.

Table 7.35. Comparison of branching strategies

data set		P0/R3/T0	P1/R3/T0	P1/R3/T1	P0/R4/T0	P0/RSP/T0
av.cpu	PRO-LB22	2.75	2.34	1.05	4.01	15.80
	PRO-DFSL	5.04	3.99	2.04	4.05	16.10
	PRO-DFSB	5.02	3.84	2.04	4.39	17.59
max.cpu	PRO-LB22	167.36	138.09	84.42	360.76	308.42
	PRO-DFSL	310.67	186.82	119.96	459.32	313.71
	PRO-DFSB	335.96	204.08	130.41	491.71	333.73
av.nodes	PRO-LB22	4,026	3,453	1,583	5,277	18,963
	PRO-DFSL	7,164	5,750	3,622	5,214	19,427
	PRO-DFSB	15,359	11,956	6,069	12,751	46,216
av.stored	PRO-LB22	964	855	446	1,936	4,623
	PRO-DFSL	1,645	1,294	727	1,948	4,717
	PRO-DFSB	1,498	1,131	684	1,912	4,723

The results of Table 7.35 show that PRO-LB22 (local lower bound method) comes out best with respect to all measures. For the data sets with resource profile R3, it is about twice as fast as PRO-DFSL and PRO-DFSB, respectively. This indicates that the local lower bound method is very successful in directing the search process. Comparing PRO-LB22 and PRO-DFSL proves that it pays off to spend effort in branching subproblems (nodes) in an advantageous sequence. Considering PRO-DFSB shows that determining such a sequence must be done efficiently in order to really obtain the acceleration. In particular, this becomes apparent when comparing the average number of subproblems constructed by PRO-LB22 and PRO-DFSB. For all the data sets, PRO-DFSB considers a multiple

of the subproblems built by PRO-LB22. This underlines that the effort of determining all subproblems of a node and applying the bound arguments before branching the first one does not result in determining good or optimal solutions earlier. Furthermore, the fact that the average number of non-dominated partial schedules stored is nearly equal indicates that many of the nodes constructed by PRO-DFSB and contained in local candidate lists can be fathomed due to their lower bound values after finding good or optimal solutions. These nodes would not have been built by PRO-LB23 at all.

Schedule Storing Rules. As already mentioned in the description of the schedule storing rules in Section 6.2.4.4, pp. 232, all present proposals for dominance rules based on comparing partial schedules restrict to partial schedules considering the same set of jobs. Therefore, we examine whether the effort caused by applying the extended schedule storing rule is justified or not. For this purpose, we compare three different versions of PROGRESS which all apply the local lower bound method but exclude the bound arguments presented in Section 6.2.3. The first version PRO- ζ_0 does not apply the extended schedule storing rule at all. The two other versions PRO- ζ_2 and PRO- ζ_4 apply the rule with parameter ζ set to 2 and 4, respectively. That is, PRO- ζ_2 equals the version PRO-LLB used in the tests so far.

Table 7.36. Analyzing the extended schedule storing rule

data set		P0/R3/T0	P1/R3/T0	P1/R3/T1	P0/R4/T0	P0/RSP/T0
av.cpu	PRO- ζ_0	3.49	2.82	1.32	5.47	19.84
	PRO- ζ_2	2.75	2.29	1.12	4.77	14.72
	PRO- ζ_4	2.88	2.38	1.11	5.32	15.25
max.cpu	PRO- ζ_0	174.28	130.80	79.47	459.26	326.51
	PRO- ζ_2	148.97	123.02	82.90	363.69	272.83
	PRO- ζ_4	158.07	129.18	79.54	424.31	290.01
av.nodes	PRO- ζ_0	6,260	5,152	2,404	10,453	28,376
	PRO- ζ_2	4,712	3,965	1,887	8,356	20,192
	PRO- ζ_4	4,689	3,951	1,880	8,345	20,110
av.stored	PRO- ζ_0	1,433	1,148	633	3,278	6,319
	PRO- ζ_2	1,133	956	522	2,784	4,813
	PRO- ζ_4	1,124	951	519	2,779	4,784

The results given in Table 7.36 reveal that applying the extended schedule storing rule succeeds in reducing the average computation times as well as the

maximum ones. It has to be pointed out that this is particularly true for the hardest of all data sets P0/RSP/T0 for which PRO- ζ_2 clearly outperforms PRO- ζ_0 . Furthermore, it can be recognized that extending the parameter ζ beyond a certain value does not pay off. This is confirmed by comparing the computation times as well as the average number of nodes which have to be branched by PRO- ζ_2 and PRO- ζ_4 , respectively.

Finally, we want to briefly address another feature which greatly helps to increase the effectiveness of the schedule storing rules. It consists of deleting stored partial schedules which are dominated by the ones currently considered. For example, in case of the most challenging data set P0/RSP/T0, the average number of partial schedules deleted when applying PRO- ζ_2 is 10,802. That is, without this feature the average number of stored partial schedules is 15,615 instead of 4,813 as given in Table 7.36. Besides the computer memory occupied by storing the data, the effort for retrieving partial schedules for dominance purposes is much higher.

7.5.2 Scattered Branch and Bound

In the sequel, we examine how the computational efficiency of PROGRESS can be increased by incorporating scattered branch and bound. Since the resulting procedure SCATTER clearly outperforms PROGRESS, in particular in determining good solutions early in the search process, it is furthermore compared to specialized RCPSP procedures as well as to the tabu search procedure RETAPS.

7.5.2.1 Comparing SCATTER to PROGRESS

In the following, we analyze the effectiveness of scattered branch and bound by applying the procedures PROGRESS and SCATTER to the GRCPSP data sets described in Section 7.2.3. For PROGRESS, it has been shown in Section 7.5.1.2 that, considering all data sets, the computational effort caused by the bounding rules is not compensated by their success in fathoming nodes. Therefore, the lower bound arguments are omitted for this test. By the way of contrast, all reduction and dominance rules are applied. For SCATTER, we choose $\mu = 100$ and $v = 25$, i.e., among 100 solutions randomly generated the best 25 are selected as cut solutions. Initially, 100 nodes of each subtree are evaluated ($\gamma = 100$) (cf. Section 6.3.2.2, pp. 248). The minimum number of nodes which have to be ex-

amined before applying a break-off rule is set to $\delta = 200$ (cf. Section 6.3.2.3, pp. 249).

A summary of the results is given in the following tables. Note that all instances are solved to optimality, i.e., the computation time is not limited for both procedures. In addition to the performance measures already examined earlier we introduce the following one:

- av.opt: average number of nodes branched per instance before finding an optimal solution

SCATTER outperforms PROGRESS with respect to all measures and all data sets except for the data set P1/R3/T1 (cf. Tables 7.37 and 7.38). Due to the time windows considered in this data set, randomly choosing a branch to be developed in each node may not lead to feasible cut solutions. In this case, no favorable partition of the enumeration tree is obtained and eventually more evolved concepts for generating cut solutions are required.

Table 7.37. Comparing PROGRESS to SCATTER for project networks P0

data set		R0/T0	R1/T0	R2/T0	R3/T0	R4/T0	RSP/T0
av.cpu	PROGRESS	0.30	0.31	0.82	2.75	4.77	14.72
	SCATTER	0.32	0.32	0.58	2.25	4.34	13.06
max.cpu	PROGRESS	42.19	42.76	60.28	148.97	363.69	272.83
	SCATTER	4.68	4.73	20.38	115.44	305.73	270.08
av.nodes	PROGRESS	618	627	1545	4 712	8 356	20 192
	SCATTER	174	177	589	3 206	7 152	16 661
av.stored	PROGRESS	225	230	478	1 133	2 784	4 813
	SCATTER	81	82	246	976	2 653	4 612
av.opt	PROGRESS	604	613	1 515	4 391	5 868	13 906
	SCATTER	147	148	540	2 859	4 025	9 915

Considering the average computation times, the decrease obtained by SCATTER does not seem to be convincing. This impression is revised when comparing the maximum computation times. For the data sets P0/R0/T0 and P0/R1/T0, a reduction of nearly 90% is yielded. Probably, the measure av.cpu does not properly reflect the benefits of scattered branch and bound. This is due to the data sets containing a large of number of instances which can be solved very efficiently by both procedures resulting in very small enumeration trees. If instances are easy to solve, scattered branch and bound can even lead to a computa-

Table 7.38. Comparing PROGRESS to SCATTER for project networks P1

data set		R3/T0	R3/T1	R4/T0	RSP/T0
av.cpu	PROGRESS	2.29	1.12	3.31	10.03
	SCATTER	1.82	1.27	3.23	9.82
max.cpu	PROGRESS	123.02	82.90	310.16	156.40
	SCATTER	92.76	72.50	243.24	153.26
av.nodes	PROGRESS	3 965	1 887	5 860	14 137
	SCATTER	2 603	1 733	5 543	12 814
av.stored	PROGRESS	956	522	1 985	3 523
	SCATTER	819	518	1 883	3 460
av.opt	PROGRESS	3 788	1 742	4 533	9 694
	SCATTER	2 379	1 558	3 817	8 443

tional overhead. Therefore, in order to get a better impression, we consider a subset of 20 hard instances for each data set. Note that these 20 instances have been derived from the same 20 SMFF instances for all data sets which allows for a more sound comparison. The results are given in Tables 7.39 and 7.40.

We recognize, that the average computation times required for solving the hard instances of the different data sets are considerably smaller for SCATTER than for PROGRESS. For the hardest of all data sets (P0/RSP/T0), applying scattered branch and bound leads to a reduction in the average computation time of nearly 40%. This is also reflected by the reduction in the average number of nodes which have to be branched for solving an instance. In general, considering a smaller number of nodes allows for applying more complex bounding and

Table 7.39. Comparing PROGRESS to SCATTER for P0 (hard instances)

data set		R0/T0	R1/T0	R2/T0	R3/T0	R4/T0	RSP/T0
av.cpu	PROGRESS	1.68	1.81	6.65	23.79	50.94	104.33
	SCATTER	1.19	1.20	3.98	17.38	41.82	66.87
max.cpu	PROGRESS	20.07	19.97	33.76	139.35	363.69	223.43
	SCATTER	4.68	4.73	20.38	94.42	305.73	196.58
av.nodes	PROGRESS	3 358	3 575	12 487	39 901	79 706	141 631
	SCATTER	1 546	1 565	6 010	26 816	63 448	86 876
av.stored	PROGRESS	1 390	1 488	3 758	8 759	23 967	23 981
	SCATTER	779	789	2 525	7 908	22 590	21 871
av.opt	PROGRESS	3 182	3 394	12 233	38 006	60 641	106 230
	SCATTER	1 292	1 307	5 778	25 188	36 046	43 674

dominance rules because they have to be evaluated less often. By the way of contrast, the number of partial schedules which have to be stored for applying the schedule storing rules does not differ considerably for the hard data sets. This indicates that for each instance there exists a basic set of partial schedules which do not dominate each other and which have to be developed in any case. However, taking into account of the average number of nodes (av.nodes), SCATTER succeeds in determining these partial schedules much earlier than PROGRESS and, thus, can utilize them for dominance purposes.

Table 7.40. Comparing PROGRESS to SCATTER for P1 (hard instances)

data set		R3/T0	R3/T1	R4/T0	RSP/T0
av.cpu	PROGRESS	20.54	8.07	32.68	60.64
	SCATTER	14.81	8.91	28.37	51.40
max.cpu	PROGRESS	102.08	38.20	310.16	148.52
	SCATTER	92.76	58.17	243.24	134.59
av.nodes	PROGRESS	34 835	13 412	51 511	84 064
	SCATTER	23 371	13 905	43 402	69 219
av.stored	PROGRESS	7 414	3 375	16 046	15 948
	SCATTER	6 277	4 220	15 210	15 944
av.opt	PROGRESS	33 693	12 474	42 129	64 667
	SCATTER	22 262	13 302	34 066	48 426

The most striking difference between SCATTER and PROGRESS can be observed when considering the average number of nodes av.opt necessary for determining an optimal solution. For the data sets P0/R0/T0 and P0/R1/T0, SCATTER finds the optimum after examining one third of the nodes which have to be evaluated by PROGRESS. Note that for the most data sets PROGRESS even requires more nodes for finding the optimal solutions than SCATTER for solving the instances completely. This result is particularly important taking into account that branch and bound procedures often have to be terminated prematurely due to limited computation time. If the time limit is exceeded, the objective function value of the best known solution provides an upper bound. Therefore, we perform a further comparison of SCATTER and PROGRESS determining the average and maximum relative deviations from optimality for different time limits (1, 5, 10, and 60 seconds). As before, relative deviations are computed by $(UB - CT)/CT \cdot 100\%$ with CT denoting the optimal project completion time of an instance. Tables 7.41 and 7.42 show the results for the hard instances.

Table 7.41. Average deviations from optimality (hard instances)

data set	1 s		5 s		10 s		60 s	
	PRO.	SCA.	PRO.	SCA.	PRO.	SCA.	PRO.	SCA.
P0/R0/T0	13.87	13.64	2.93	0.00	2.67	0.00	0.00	0.00
P0/R1/T0	13.84	13.64	4.88	0.00	2.67	0.00	0.00	0.00
P0/R2/T0	18.21	19.14	14.72	8.14	9.68	4.40	0.00	0.00
P0/R3/T0	20.23	19.34	13.69	13.60	12.46	10.48	4.71	2.94
P0/R4/T0	4.02	3.02	2.52	1.38	1.74	0.94	0.55	0.16
P0/RSP/T0	10.92	8.44	7.78	3.46	6.46	2.02	1.98	0.29
P1/R3/T0	27.28	22.98	20.33	15.76	14.24	11.16	1.72	1.72
P1/R3/T1	8.69	25.10	10.86	15.47	8.09	7.58	0.00	0.00
P1/R4/T0	3.71	3.10	2.33	1.26	1.14	0.65	0.24	0.16
P1/RSP/T0	10.04	8.51	6.29	4.74	4.77	3.27	0.61	0.54

Only for a time limit of 1 second, the performance of both procedures is nearly identical. Allowing 5 seconds of computation time, SCATTER has already determined an optimal solution for each hard instance of the data sets P0/R0/T0 and P0/R1/T0 whereas the average deviation of the solutions found by PROGRESS is nearly 3% and 5%, respectively. In both cases, the maximum deviation is even 58.62%. For the other data sets, the average as well as maximum deviations yielded by SCATTER are much smaller, too. Also considering larger time limits, SCATTER outperforms PROGRESS. For the most challenging data set P0/RSP/T0,

Table 7.42. Maximum deviations from optimality (hard instances)

data set	1 s		5 s		10 s		60 s	
	PRO.	SCA.	PRO.	SCA.	PRO.	SCA.	PRO.	SCA.
P0/R0/T0	64.71	60.20	58.62	0.00	53.45	0.00	0.00	0.00
P0/R1/T0	63.79	60.20	58.62	0.00	53.45	0.00	0.00	0.00
P0/R2/T0	64.18	61.22	58.62	50.00	55.22	31.43	0.00	0.00
P0/R3/T0	54.41	46.75	45.59	40.26	42.65	40.26	34.33	27.59
P0/R4/T0	16.18	9.78	13.24	5.43	10.29	5.17	4.48	1.72
P0/RSP/T0	16.54	15.25	13.11	10.59	13.11	10.59	9.84	2.46
P1/R3/T0	55.88	49.35	51.95	45.45	48.05	44.16	34.33	34.28
P1/R3/T1	100.00	97.40	100.00	96.23	40.30	40.30	0.00	0.00
P1/R4/T0	16.18	10.87	13.24	6.49	7.46	3.45	2.99	1.72
P1/RSP/T0	22.95	21.31	14.75	14.75	13.11	13.11	6.56	6.56

T0, PROGRESS requires 60 seconds to achieve results comparable to those of SCATTER for a time limit of 10 seconds. The only exception for small time limits again is the data set P1/R3/T1 due to the reasons stated at the beginning of this section.

The test reveals some further interesting insight. Though the data sets with rather restrictive resource profiles (R0 to R3) are much easier to solve than those with less restrictive ones (R4, RSP), yielding solutions with small deviations from optimality is harder. That is, for these restrictive resource profiles the difficulty consists more in finding the optimal solutions than in proving their optimality. Note that this observation has already been made for the tabu search procedure RETAPS in Section 7.4.2.1.

7.5.2.2 Comparing SCATTER to Existing RCPSP Procedures

In the sequel, we compare SCATTER to specialized exact RCPSP procedures to examine its efficiency despite the computational overhead caused by handling minimum time lags and varying resource profiles. For this comparison, we use the version of SCATTER as described earlier except for now including the bounding rules LBC2' and LBM2', because these rules have turned out to be effective when considering non-varying resource profiles (cf. Section 7.5.1.2, pp. 309). As reference procedures those proposed by Demeulemeester and Herroelen (1997 b), Sprecher (1997), and Brucker et al. (1998) called DH, GSA, and BKST for short, are chosen (cf. Section 6.4, pp. 252).

First of all, a comparison is made for the standard SMFF data set. Additionally, we consider the more challenging data set J60 for some instances of which optimal solutions are still unknown. Hence, it represents a good test bed for analyzing the efficiency of SCATTER in finding near-optimal solutions early. For this data set, computational results are only available for the procedures BKST and GSA. Note that the results refer to the complete SMFF (P0/R4/T0) and J60 data set, respectively, i.e., containing all 480 instances even the trivial ones with a resource strength of RS = 1.0.

Unfortunately, the computational results for the four different procedures SCATTER, BKST, DH, and GSA have been obtained on different computer platforms. BKST is tested using a SUN/Sparc 20/801 workstation with a clockpulse of 80 MHz running the operating system Solaris 2.5. DH is executed on a computer with an Intel 486 processor and a clockpulse of 25 MHz using the operating

system Windows NT 3.51. Finally, the results for GSA concerning the data set SMFF are yielded on a computer with an Intel 486 processor and 66 MHz clockpulse running the operating system LINUX whereas those for J60 are determined on a Intel Pentium computer with 166 MHz clockpulse also using LINUX.

Data Set SMFF. For this data set, we compare the average and maximum computation times which are required by SCATTER, DH, and GSA for finding and proving optimal solutions for all instances. BKST is excluded from this comparison, because it represents the only procedure which has not been able to solve all instances to optimality within reasonable time. Considering a time limit of 1 hour per instance, optimal solutions have been yielded only for 425 of the 480 instances. For the remaining procedures, a very rough comparison is only possible based on factors describing the gain of speed obtained by using faster processors in comparison to the slowest one, the Intel 486 with 25 MHz clockpulse. The speed up factors we have used are 2.64 for the 486 processor with 66 MHz clockpulse (GSA) and 8.00 for our Intel Pentium with 166 MHz, respectively.

In Table 7.43, we give converted computation times referring to the Intel 486 / 25 MHz computer. The results show that, though considering the more general GRCPSP, SCATTER performs better than the algorithms DH and GSA with the latter being clearly outperformed. This is due to the fact that GSA has been developed in order to obtain good results with low memory requirements. If only 512 KB memory are allowed for storing partial schedules in order to apply dominance rules, GSA clearly performs better than all other approaches for RCPSP (cf. Sprecher (1997)). Unfortunately, as already indicated in Remark 6.2.4.4, p. 232, many of the dominance rules contained in GSA can not be transferred to GRCPSP due to the minimum time lags and the varying resource profiles. In Herroelen et al. (1998), it is indicated (without giving details) that the efficiency of DH can be further increased by a new lower bound, a new dominance rule and a more efficient implementation.

Table 7.43. Comparing SCATTER to RCPSP procedures for SMFF

procedure	SCATTER	DH	GSA
av.cpu	24.08	33.71	115.42
max.cpu	2,886	9,515	36,038

Data Set J60. Since not all instances of the J60 data set can be solved to optimality in reasonable time, different time limits have been used when testing the procedures BKST and GSA (excluding input and output operations). When reaching a time limit, only an upper bound on the optimal project completion time is available. Furthermore, average and maximum deviations are computed referring to lower bound values instead of the (unknown) optimal project completion times. For the sake of comparability, we use the lower bound values obtained by Baar et al. (1998) which can be downloaded from the web at <http://scarlett.mathematik.uni-osnabrueck.de/research/or/rcpsp/rcpsp.html>.

A summary of the results is presented in Table 7.44. The row "time limit" denotes the time limit in seconds after which each procedure has prematurely been terminated (on the respective computer system). Note that for this comparison the computation times are not converted. Both, GSA and SCATTER are executed on nearly identical computers except for the operating system. By the way of contrast, BKST is applied on a computer platform with a completely different architecture such that determining an appropriate factor is difficult.

Table 7.44. Comparing SCATTER to RCPSP procedures for J60

	BKST	GSA		SCATTER				
time limit	3600	300	1800	10	60	300	1800	3600
#opt	326	349	364	334	352	365	385	393
av.cpu	1162.06	88.07	472.69	3.68	17.78	77.73	396.66	736.14
av.dev	4.80	5.72	5.27	5.27	4.82	4.55	4.30	4.21
max.dev	30.80	45.76	40.68	32.53	31.25	29.69	29.69	29.03

The results reveal that SCATTER clearly outperforms the procedures BKST and GSA even taking into account the different computer platforms used. Within a time limit of 10 seconds it already solves more instances to optimality than BKST in one hour of computation time (on the SUN/Sparc). This result is still clear even assuming our Pentium based computer being two to three times faster. Considering GSA (same processor, only different operating system) for 300 and 1800 seconds, more optimal solutions are found and proven by SCATTER within a time limit of 60 and 300 seconds, respectively. Furthermore, its average computation times relative to the respective time limit given are smaller than the ones required by the other procedures. Also for the measures recording the relative deviations from the lower bound values it performs better. After 60 seconds the relative deviations are approximately as large as those of BKST af-

ter one hour (on the SUN/Sparc). GSA is not able to yield smaller average and maximum deviations within a time limit of 1800 seconds than SCATTER has already obtained after 10 seconds. However, naturally for less instances the optimality has been proven by SCATTER within this short time.

The potential of SCATTER becomes even more apparent taking into account that the optimal project completion time of 296 of the 480 instances is equal to the simple lower bound value LBC1. Of the remaining challenging 184 instances SCATTER solves 97 to optimality whereas BKST and GSA find and prove optima only for 30 and 68 instances, respectively.

7.5.2.3 Comparing SCATTER to RETAPS

The computational experiments in the previous sections revealed that SCATTER is able to determine good solutions early in the search process. Therefore, it may also be promising to use a truncated version as a heuristic procedure. Similar approaches have, e.g., been examined for RCPSP by Pollack-Johnson (1995) and for RCPSP with maximum time lags by Schwindt (1998, chapter 4). Whereas in these approaches, the search strategies of the corresponding procedures are modified in order to enhance the heuristic capabilities, SCATTER is applied as described in Section 7.5.2.1, because it contains a component to diversify the search by default. Since no other efficient heuristics procedures for GRCPSP are on hand, we use the tabu search procedure RETAPS as reference. Within the experiment, the average and maximum relative deviations from optimality as well as the number of instances solved to optimality are determined for different time limits (1, 5, 10, and 60 seconds) considering the data sets introduced in Section 7.2.3.

The results obtained are summarized in the Tables 7.45 to 7.47 with the best value obtained for each time limit being typeset bold. Considering the average deviations from optimality, none of the procedures outperforms the other in general. For the very short time limit of 1 second, preference should be given to the procedure RETAPS, whereas for the rather large time limit of 60 seconds SCATTER seems to be the method of choice. Concerning the time limits in between, RETAPS nearly always yields slightly better results than SCATTER with data set P1/R3/T1 representing the only considerable exception. For this data set, the time windows defined narrow the solution space considerably, such that the systematic search performed by SCATTER turns out to be advantageous.

Table 7.45. Average deviations from optimality

data set	1 s		5 s		10 s		60 s	
	SCA.	RET.	SCA.	RET.	SCA.	RET.	SCA.	RET.
P0/R0/T0	1.68	1.37	0.00	0.08	0.00	0.00	0.00	0.00
P0/R1/T0	1.92	1.35	0.00	0.00	0.00	0.00	0.00	0.00
P0/R2/T0	3.39	1.65	1.03	0.58	0.32	0.04	0.00	0.00
P0/R3/T0	6.01	3.82	1.83	1.53	1.13	1.23	0.32	0.67
P0/R4/T0	0.69	0.36	0.20	0.11	0.13	0.06	0.01	0.01
P0/RSP/T0	4.64	2.33	1.44	1.29	0.68	0.99	0.08	0.49
P1/R3/T0	5.84	3.58	2.07	1.71	1.28	0.95	0.13	0.49
P1/R3/T1	4.63	4.66	1.69	2.80	0.80	1.58	0.09	0.72
P1/R4/T0	0.69	0.37	0.20	0.15	0.09	0.08	0.02	0.02
P1/RSP/T0	4.44	2.15	1.25	1.09	0.71	0.76	0.07	0.29

Examining the maximum deviations, a slightly different picture is obtained (cf. Table 7.46). For the rather restrictive data sets with resource profiles R0 to R3, SCATTER clearly yields the better results in particular for the time limits of 1 and 5 seconds is used. By the way of contrast, the same is true for RETAPS and the data sets with resource profiles R4 and RSP. Finally, considering the number of instances solved to optimality, no major differences in the performance of both procedures can be observed except for the data sets with resource profile RSP, where SCATTER outperforms RETAPS (Table 7.48).

Table 7.46. Maximum deviations from optimality

data set	1 s		5 s		10 s		60 s	
	SCA.	RET.	SCA.	RET.	SCA.	RET.	SCA.	RET.
P0/R0/T0	81.25	84.44	0.00	29.41	0.00	0.00	0.00	0.00
P0/R1/T0	81.25	100.00	0.00	0.00	0.00	0.00	0.00	0.00
P0/R2/T0	63.79	100.00	50.00	100.00	31.43	12.80	0.00	0.00
P0/R3/T0	84.62	92.39	40.26	92.39	40.26	92.39	27.59	92.39
P0/R4/T0	9.78	5.88	5.43	3.45	5.17	3.13	1.72	1.56
P0/RSP/T0	16.25	12.94	13.51	11.82	12.16	11.82	3.96	10.19
P1/R3/T0	49.35	100.00	45.45	100.00	44.16	100.00	34.48	100.00
P1/R3/T1	98.91	100.00	96.23	100.00	40.30	100.00	26.25	100.00
P1/R4/T0	10.87	6.58	6.49	4.41	4.84	3.17	3.23	1.56
P1/RSP/T0	21.31	13.10	14.75	9.46	14.75	7.06	6.56	6.36

In general, the observations made allow for the conclusion that none of the procedures can be preferred to the other. This is underlined when examining the results more carefully. For example for the data set P1/R3/T0, RETAPS does not manage to yield a satisfying solution quality for all instances even after 60 seconds of computation time. However, the recorded maximum deviation of 100 % is due to only two instances which can be solved to optimality by SCATTER within less than 1 second. Vice versa, the same situation occurs for some instances where large maximum deviations have to be accepted when applying SCATTER.

Table 7.47. Number of instances solved to optimality

data set	1 s		5 s		10 s		60 s	
	SCA.	RET.	SCA.	RET.	SCA.	RET.	SCA.	RET.
P0/R0/T0	349	350	360	359	360	360	360	360
P0/R1/T0	348	351	360	360	360	360	360	360
P0/R2/T0	336	347	352	357	357	359	360	360
P0/R3/T0	285	310	336	337	346	345	356	355
P0/R4/T0	284	308	327	337	335	346	355	357
P0/RSP/T0	93	150	214	208	263	238	339	279
P1/R3/T0	290	317	336	339	346	346	359	356
P1/R3/T1	315	315	345	334	351	347	359	356
P1/R4/T0	295	308	334	330	342	341	356	354
P1/RSP/T0	100	170	236	223	286	251	345	302

Unfortunately, it can not be decided in advance which procedure should be used for a specific instance. For example, no general relationship between the complexity measures proposed in Section 7.2.1 and the performance of the procedures could be detected as this is, e.g., the case for different configurations of destructive improvement (cf. Section 7.3.3). Therefore, we have examined the following approach of a combined heuristic. Both procedures are applied for each instance with an identical time limit of 5 seconds thus totalling in a maximum computation time of 10 seconds. Note that the procedures do not exchange informations, e.g., upper bound values, such that they can be executed in an arbitrary sequence. Among the objective function values computed the smaller one is selected. The results obtained concerning the measures already examined before are given in Table 7.48. For the sake of convenience, also the

values obtained by each procedure when applied for 10 seconds solely are replicated from the Tables 7.46 to 7.48.

Table 7.48. Results for combined heuristic after 10 seconds

data set	av.dev			max.dev			#opt		
	SCA.	RET.	COMB.	SCA.	RET.	COMB.	SCA.	RET.	COMB.
P0/R0/T0	0.00	0.00	0.00	0.00	0.00	0.00	360	360	360
P0/R1/T0	0.00	0.00	0.00	0.00	0.00	0.00	360	360	360
P0/R2/T0	0.32	0.04	0.00	31.43	12.80	0.00	357	359	360
P0/R3/T0	1.13	1.23	0.19	40.26	92.39	26.53	346	345	355
P0/R4/T0	0.13	0.06	0.05	5.17	3.13	3.45	335	346	351
P0/RSP/T0	0.68	0.99	0.67	12.16	11.82	10.59	263	238	276
P1/R3/T0	1.28	0.95	0.21	44.16	100.00	25.30	346	346	355
P1/R3/T1	0.80	1.58	0.25	40.30	100.00	29.17	351	347	356
P1/R4/T0	0.09	0.08	0.05	4.84	3.17	2.60	342	341	349
P1/RSP/T0	0.71	0.76	0.55	14.75	7.06	8.86	286	251	286

The results reveal that the combined heuristic clearly outperforms the single procedures with respect to all measures. This is in particular true for the data sets with resource profile R3 where the two procedures complement each other favorably. For these data sets, the average and maximum deviations from optimality can be reduced dramatically. To conclude, the performance of procedures for GRCPSP seems to be much more data dependent than, e.g., for the classical resource-constrained project scheduling problem RCPSP. Therefore, it may be appropriate to apply a variety of procedures for short time rather than restricting to a single one.

8 Summary and Conclusions

The use of project management is continuously growing in industrial and public organizations providing an efficient instrument for mastering the challenges caused by steadily shortening product life cycles, decreasing profit margins, and global markets. With the size and complexity of projects increasing, the ability to efficiently plan and control them becomes an essential part of their management. In particular, scheduling which is concerned with determining execution dates for the sub-activities (jobs) required for completing the project presents a major managerial task. This task becomes mathematically complex as soon as the limited availability of resources forces conflicts between concurrent projects or even sub-activities of a single project. Therefore, scheduling is commonly supported by project management software packages which are available in a large variety. However, the quality of the schedules obtained by the included solution methods is often rather poor. The book on hand deals with decision problems which arise in this context (Part I). Modern computer-based procedures which can be applied for solving these problems are described in Part II. To examine their effectiveness comprehensive computational experiments are performed and reported.

After defining the term "project", Chapter 1 gives a basic outline on the management process which accompanies a project from its initiation until its completion. This outline is organized along the project life cycle which is usually subdivided into five major phases. Each of these phases is discussed separately. Within the respective sections, the arising project management tasks are identified and possibilities for accomplishing them are briefly discussed.

In Chapter 2, basic methods and tools of project planning and control are described. In general, project planning consists of establishing a schedule and a budget for executing the project. The description follows the typical planning process starting with structuring the project. In Section 2.1, methods and tools usually employed for this purpose are presented. This includes the work breakdown structure identifying the jobs which have to be executed in order to com-

plete the project as well as activity-on-node and activity-on-arc diagrams which graphically depict precedence relationships between the jobs. Section 2.2 is devoted to simple schedule computations. By application of the critical path method, the smallest project completion time as well as time windows for the execution of jobs are determined. Furthermore, Gantt charts are commonly used to communicate the resulting schedule. Compromising the next step in the planning process, the topic of resource allocation is discussed in Section 2.3. The basic tools used in this context, i.e., the resource loading profile as well as the capacity-oriented Gantt chart, are introduced. For each period of the considered planning horizon, these tools allow to visually compare the availability of a resource type to its demand caused by the jobs currently processed. Subsequently, two basic methods for resolving possible resource conflicts occurring if in any period the demand of a resource type exceeds the availability are outlined. Within resource-constrained scheduling, the starting times of jobs are rearranged such that the project is completed as early as possible and all resource conflicts are removed. If the maximum project completion time is limited, time-constrained scheduling aims at determining starting times of jobs such that the cost for providing additional resources are minimized.

Section 2.4 is devoted to project control, representing the major management task during the execution of a project. For schedule control, which ensures that the project is finished on time, a variety of tools is presented. Among these are the progress Gantt chart, the progress plot, and the trend analysis. For cost control which verifies whether the project is performed on budget, the well-known earned value analysis is described. Finally, Section 2.5 contains a survey on the major features of project management software packages supporting the planning and control process.

In Chapter 3, decision problems arising in the context of resource-constrained project scheduling are discussed in detail based on definitions and notations given in Section 3.1. First of all, Section 3.2 deals with the classical resource-constrained project scheduling problem (RCPSP). Different mathematical formulations which have been proposed for this problem in the literature are presented. Furthermore, new formulations are developed which, e.g., can be applied for analyzing resource-constrained schedules. Finally, a generalized problem version GRCPSP is introduced. Among others, the most important generalization concerns the circumstance that also fluctuations in the resource availabilities, e.g., due to maintenance of equipment or vacations, may be taken

into account. In addition to those generalizations already provided by GRCPSP, further ones may have to be incorporated in order to describe a particular planning situation appropriately. A survey on those generalizations which have been described in the literature so far is given in Section 3.3. However, the generalized problems discussed all have in common that they share the objective function of minimizing the project completion time. Further decision problems may arise when alternative, e.g., cost-related objective functions have to be considered. Such decision problems are reviewed in Section 3.4.

Part II of this book is devoted to solution methods for resource-constrained project scheduling. Chapter 4 deals with possibilities for computing lower bounds on the smallest project completion time of RCPSP and GRCPSP instances.

In Section 4.1, constructive lower bound arguments for RCPSP are described which directly compute a bound value by relaxing the problem and, subsequently, solving the relaxation. Two basic types of lower bound arguments can be distinguished. Simple bound arguments can be realized with small computational effort. In the main, they are based on relationships of RCPSP to other combinatorial optimization problems such as bin packing or specialized machine scheduling problems. Besides describing existing arguments of this type, a number of new ones is introduced. Subsequently, complex bound arguments are surveyed. Among these are linear programming based approaches and Lagrangean relaxation.

In Section 4.2, a new meta-strategy for computing lower bounds (on minimization problems), called destructive improvement, is proposed. The basic idea of this approach is to prove that no feasible solution for a trial bound value can exist. In case of success, the trial value may be increased and the process is repeated for a larger value. Two basic steps are performed to furnish the proof. Assuming that the objective function value must not exceed the trial one, the problem data is reduced. If this does not result in a contradiction, modified constructive bounding procedures, taking advantage of the reduced data, are computed. In order to apply destructive improvement to RCPSP, different reduction techniques are introduced as well as the simple bound arguments are adapted. Finally, Section 4.3 describes how the bound arguments introduced for RCPSP can be adjusted such that they can also be employed for GRCPSP.

Chapter 5 is concerned with heuristic procedures for solving RCPSP and GRCPSP. In order to ease their presentation, different types of schedules are introduced in Section 5.1. Most of the procedures proposed fall into one of the following two categories. Priority-rule based heuristics, which are presented in Section 5.2, apply some scheduling scheme in order to construct a single schedule. While constructing this schedule, the jobs are considered according to priority values. After discussing the two basic scheduling schemes (serial and parallel) available, these are extended for RCPSP by incorporating backward and bidirectional planning. Subsequently, a survey on priority rules for RCPSP is given and the most effective ones are adapted to work with GRCPSP. Finally, multi-pass heuristics are considered which determine a set of solutions by applying several different or randomized priority rules.

The other category of procedures presented in Section 5.3 is based on some meta-heuristic approach. Basically, a meta-heuristic can be defined as a general strategy for guiding other heuristics such that solutions beyond locally optimal ones are obtained. Among the most popular meta-heuristics are simulated annealing, genetic algorithms, and tabu search. After giving an introduction into the latter one, a new tabu search procedure, called RETAPS, for solving GRCPSP is proposed. Finally, an overview on existing meta-heuristic based procedures for RCPSP is presented.

Chapter 6 concentrates on procedures for exactly solving RCPSP and GRCPSP. Nearly all approaches of this type which have been presented so far rely on the branch and bound principle which is introduced in Section 6.1. Section 6.2 describes a new branch and bound procedure for GRCPSP, called PROGRESS. One of its main features is the efficient combination of a parallel branching scheme and a new type of a depth-first search strategy, the local lower bound method, which has already been applied successfully to simple assembly line balancing and bin packing problems by the author. Furthermore, a number of new and improved reduction and dominance rules are proposed.

In section 6.3, a new strategy, called scattered branch and bound, for improving the performance of branch and bound procedures is proposed. The basic idea consists of decomposing the potential enumeration tree into different disjoint subtrees and evaluating these subtrees independently with the most promising ones being considered first. Additionally, depending on the development of the search process, the search can be diversified by changing to another subtree or intensified by continuing with examining the current one. After describing the

basic principles of scattered branch and bound, a corresponding modification of PROGRESS, called SCATTER, is developed. Finally, a survey on existing branch and bound procedures for RCPSP and GRCPSP is provided.

Chapter 7 reports on comprehensive computational experiments which have been performed in order to evaluate the effectiveness of the procedures described in this book. The software and hardware environment in which the new procedures have been implemented and tested are described in Section 7.1.

In Section 7.2, the test bed which has been used within the computational studies is defined. For RCPSP, a number of benchmark data sets exist the most challenging ones of which have been generated systematically. The corresponding generation process is controlled by complexity measures which aim at predicting the complexity of instances in terms of computation time. After introducing the according measures, the RCPSP benchmark data sets are described. By way of contrast, no challenging standard benchmark data sets have been established for GRCPSP in the literature so far. Therefore, new ones have been generated by the author. A description of the generation process is given at the end of the section.

Section 7.3 examines the effectiveness of lower bound arguments for RCPSP. The computational experiments show that the new simple bound arguments clearly outperform the ones described in the literature so far. The results obtained can be further improved by applying the meta-strategy destructive improvement. In fact, it turns out that for some problem classes the direct bound arguments may even be omitted completely, i.e., applying only reduction techniques, without a considerable loss in bound quality. Finally, a comparison between destructive improvement and complex bound arguments is performed showing that the destructive improvement approach can cope with these arguments except for a very recent procedure. This procedure representing an advancement of the ideas presented in this book combines destructive improvement with a complex bound argument.

Section 7.4 is devoted to computational experiments evaluating the performance of heuristic procedures. First of all, the results obtained for RCPSP reveal that bidirectional planning clearly outperforms unidirectional one when applied within single-pass priority-rule based heuristics. By combining both scheduling schemes proposed with a subset of successful priority rules and employing unidirectional and bidirectional planning, very efficient multi-pass heu-

istics can be designed. They can even cope with more versatile adaptive sampling procedures when the number of schedules to be constructed is restricted. Comparing their performance to those of proprietary heuristics of commercial software packages underlines the need of having efficient, transparent and easy to implement heuristics on hand. However, the results yielded for GRCPSP by priority-rule based heuristics are rather discouraging which is due to the complexity inherent to this problem. They can considerably be improved by applying the tabu search procedure RETAPS. Furthermore, it is shown that though RETAPS has been developed for GRCPSP it is one of the most efficient meta-heuristic based procedures for solving RCPSP.

Finally, Section 7.5 describes computational studies concerning exact procedures for RCPSP and GRCPSP. In a first step, the branch and bound procedure PROGRESS is compared to the most efficient approach for GRCPSP currently on hand. The results show that PROGRESS clearly comes out first. An in-depth analysis indicates that this is mainly due to the local lower bound method as well as the new dominance rules. Subsequently, the scattered branch and bound version SCATTER is examined revealing that it yields considerably better results than its ancestor. Furthermore, for the most challenging data set available for comparing exact RCPSP procedures, it outperforms those special purpose procedures which have been applied to this data set though it is designed for the more general problem. Finally, by combining RETAPS and a truncated version of SCATTER, a very efficient heuristic procedure can be obtained.

In general, the following conclusions can be drawn from the research reported in this book representing possible points of departure for future research. From the algorithmic point of view, a number of highly problem specific improvements concerning lower bound arguments as well as heuristic and exact procedures for RCPSP and GRCPSP have been presented. However, also two more general concepts, namely the destructive improvement technique and scattered branch and bound, have been introduced. In order to examine the potential of these approaches, they have to be applied to other capacity constrained optimization problems. For destructive improvement, the first successful applications to other problems in the field of project and job-shop scheduling have already been reported. Concerning scattered branch and bound, the results obtained indicate that it may be successfully employed in particular when problems are considered for which effective bounding and dominance rules are on hand.

From the resource-constrained project scheduling point of view, a number of challenges remain. The decision problems for which solution procedures exist are still rather restrictive allowing to represent real-world planning situations only partially. Thus, possible extensions as well as appropriate solution approaches have to be developed. Furthermore, the effectiveness of the procedures proposed so far has only been examined for problem instances of rather small size concerning, e.g., the number of jobs and resource types. Therefore, it has to be verified whether the results obtained are still valid for large-scale problems or whether new solution concepts have to be examined. Finally, the solution procedures have to be made available to practitioners by integrating them into standard software for project scheduling.

References

- Aarts, E. and J. Korst (1989): Simulated Annealing and Boltzmann Machines. Wiley, Chichester.
- Aarts, E. and J.K. Lenstra (Eds.) (1997): Local Search in Combinatorial Optimization. Wiley, Chichester.
- Abdul-Kadir, M.R. and A.D.F. Price (1995): Conceptual Phase of Construction Projects. International Journal of Project Management 6, 387-393.
- Agarwal, M.K.; S.E. Elmaghraby, and W.S. Herroelen (1996): DAGEN: A Generator of Testsets for Project Activity Nets. European Journal of Operational Research 90, 376-382.
- Ahn, T. and S. Selcuk Erenguc (1998): The Resource Constrained Project Scheduling Problem with Multiple Crashable Modes: A Heuristic Procedure. European Journal of Operational Research 107, 250-259.
- Ahuja, H.N. (1976): Construction Performance Control by Networks. Wiley, New York.
- Alexander, G.J. and W.F. Sharpe (1989): Fundamentals of Investment. Englewood Cliffs, New Jersey.
- Alvarez-Valdés, R. and J.M. Tamarit (1989 a): Algoritmos Heurísticos Deterministas y Aleatorios en Sequenciación de Proyectos con Recursos Limitados. Questiío 13, 173-191.
- Alvarez-Valdés, R. and J.M. Tamarit (1989 b): Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis. In: Slowinski, R. and J. Weglarz (Eds.): Advances in Project Scheduling. Elsevier, Amsterdam, 113-134.
- Alvarez-Valdés, R. and J.M. Tamarit (1993): The Project Scheduling Polyhedron: Dimension, Facets and Lifting Theorems. European Journal of Operational Research 67, 204-220.
- Andersen, E.S. (1996): Warning: Activity Planning Is Hazardous to Your Project's Health. International Journal of Project Management 14, 89-94.
- Angus, R.B. and N.A. Gunderson (1997): Planning, Performing, and Controlling Projects – Principles and Applications. Prentice-Hall, London.
- Archibald, R.D. (1993): Project Team Planning: A Strategy for Success. In: Dinsmore, P.C. (Ed.): The AMA Handbook of Project Management. Amacom, New York, 71-78.
- Ashok Kumar, V.K. and C. Rajendran (1989): Manpower Resource Leveling in the Maintenance of a Drillship: A Computer Simulation Approach. International Journal of Modelling & Simulation 13, 152-155.
- Assad, A.A. and E.A. Wasil (1986): Project Management Using a Microcomputer. Computers & Operations Research 13, 231-260.
- Au, T. (1988): Profit Measures and Methods of Economic Analysis for Capital Project Selection. Journal of Management in Engineering 4, 217-228.

- Baar, T.; P. Brucker, and S. Knust (1998): Tabu-Search Algorithms for the Resource-Constrained Project Scheduling Problem.** In: Voss S.; S. Martello, I. Osman, and C. Roucairol (Eds.): *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Dordrecht, 1-18.
- Badiru, A.B. and P.S. Pulat (1995): Comprehensive Project Management: Integrating Optimization Models, Management Principles, and Computers.** Prentice Hall, Englewood Cliffs.
- Baker, B.M. (1997): Cost/Time Trade-Off Analysis for the Critical Path Method: A Derivation of the Network Flow Approach.** Journal of the Operational Research Society 48, 1241-1244.
- Baker, K.R. (1974): Introduction to Sequencing and Scheduling.** Wiley, New York.
- Balachandra, R. and J. A. Raelin (1980): How to Decide When to Abandon a Project.** Research Management 23, 24-29.
- Balas, E. (1970): Project Scheduling with Resource Constraints.** In: Beale, E.M.L. (Ed.): *Applications of Mathematical Programming Techniques*. American Elsevier, New York, 187-200.
- Bandelloni, M.; M. Tucci, and R. Rinaldi (1994): Optimal Resource Leveling Using Non-Serial Dynamic Programming.** European Journal of Operational Research 78, 162-177.
- Bard, J.F.; R. Balachandra, and P.E. Kaufmann (1988): An Interactive Approach to R&D Project Selection and Termination.** IEEE Transactions on Engineering Management 35, 139-146.
- Baroum, S.M and J.H. Patterson (1999): An Exact Solution Procedure for Maximizing the Net Present Value.** In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 107-134.
- Bartusch, M.; R.H. Möhring, and F.J. Radermacher (1988): Scheduling Projects with Resource Constraints and Time Windows.** Annals of Operations Research 16, 201-240.
- Battersby, A. (1967): Network Analysis.** Macmillan, New York.
- Battiti, R. (1996): Reactive Search: Toward Self-Tuning Heuristics.** In: Rayward-Smith, V.J.; I.H. Osman, C.R. Reeves, and G.D. Smith (Eds.): *Modern Heuristic Search Methods*. Wiley, Chichester, 61-83.
- Battiti, R. and G. Tecchiolli (1994 a): Simulated Annealing and Tabu Search in the Long Run: A Comparison on QAP Tasks.** Computer and Mathematics with Applications 28, 1-8.
- Battiti, R. and G. Tecchiolli (1994 b): The Reactive Tabu Search.** ORSA Journal on Computing 6, 126-140.
- Battiti, R. and G. Tecchiolli (1995): Local Search with Memory: Benchmarking RTS.** OR Spektrum 17, 67-86.
- Bedworth, D.D. (1973): Industrial Systems: Planning, Analysis, Control.** Ronald Press, New York.
- Bell, C.E. and J. Han (1991): A New Heuristic Solution Method in Resource-Constrained Project Scheduling.** Naval Research Logistics 38, 315-331.

- Bell, C.E. and K. Park (1990): Solving Resource-Constrained Project Scheduling Problems by A* Search. *Naval Research Logistics* 37, 61-84.
- Bendell, A.; D. Solomon, and J.M. Carter (1995): Evaluating Project Completion Times When Activity Times Are Erlang Distributed. *Journal of the Operational Research Society* 46, 867-882.
- Berger, I.; J.-M. Bourjolly, and G. Laporte (1992): Branch-and-Bound Algorithms for the Multi-Product Assembly Line Balancing Problem. *European Journal of Operational Research* 58, 215-222.
- Bey, R.B.; R.H. Doersch, and J.H. Patterson (1981): The Net Present Value Criterion: Its Impact on Project Scheduling. *Project Management Quarterly* 12, Issue 2, 35-45.
- Bianco, L.; M. Caramia, and P. Dell'Olmo (1999): Solving a Preemptive Scheduling Problem with Coloring Techniques. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 1-27.
- Bianco, L.; P. Dell'Olmo, and M. Grazia Speranza (1998): Heuristics for Multi-Mode Scheduling Problems with Dedicated Resources. *European Journal of Operational Research* 107, 260-271.
- Blazewicz, J.; J.K. Lenstra, and A.H.G. Rinnooy Kan (1983): Scheduling Subject to Resource Constraints: Classification and Complexity. *Discrete Applied Mathematics* 5, 11-24.
- Blazewicz, J.; W. Domschke, and E. Pesch (1996): The Job Shop Scheduling Problem: Conventional and New Solution Techniques. *European Journal of Operational Research* 93, 1-33.
- Bock, D.B. and J.H. Patterson (1990): A Comparison of Due Date Setting, Resource Assignment, and Job Preemption Heuristics for the Multiproject Scheduling Problem. *Decision Sciences* 21, 387-402.
- Boctor, F.F. (1990): Some Efficient Multi-Heuristic Procedures for Resource-Constrained Project Scheduling. *European Journal of Operational Research* 49, 3-13.
- Boctor, F.F. (1993): Heuristics for Scheduling Projects with Resource Restrictions and Several Resource-Duration Modes. *International Journal of Production Research* 31, 2547-2558.
- Boctor, F.F. (1996 a): A New and Efficient Heuristic for Scheduling Projects with Resource Restrictions and Multiple Execution Modes. *European Journal of Operational Research* 90, 349-361.
- Boctor, F.F. (1996 b): Resource-Constrained Project Scheduling by Simulated Annealing. *International Journal of Production Research* 34, 2335-2351.
- Boehm, B.W. (1981): *Software Engineering Economics*. Prentice Hall, Englewood Cliffs.
- Boehm, B.W. (1988): A Spiral Model of Software Development and Enhancement. *Computer*, May Issue, 61-72.
- Borland (1996): *Borland C++ – Programmierhandbuch*. Borland GmbH, Langen.
- Böttcher, J.; A. Drexl, R. Kolisch, and F. Salewski (1999): Project Scheduling under Partially Renewable Resource Constraints. *Management Science* 45, 543-559.

- Bouleimen, K. and H. Lecocq (1998): A New Efficient Simulated Annealing Algorithm for the Resource-Constrained Project Scheduling Problem. Working Paper, University of Liège.
- Bowers, J.A. (1995): Criticality in Resource Constrained Networks. *Journal of the Operational Research Society* 46, 80-91.
- Bowman, E.H. (1959): The Schedule-Sequencing Problem. *Operations Research* 7, 621-624.
- Brand, J.D.; W.L. Meyer, and L.R. Shaffer (1964): The Resource Scheduling Method for Construction. *Civil Engineering Studies Report No. 5*, University of Illinois.
- Brinkmann, K. and K. Neumann (1996): Heuristic Procedures for Resource-Constrained Project Scheduling with Minimal and Maximal Time Lags: The Resource-Levelling and Minimum Project Duration Problems. *Journal of Decision Systems* 5, 129-155.
- Brooks, G.H. and C.R. White (1965): An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem. *Journal of Industrial Engineering*, January-February Issue, 34-40.
- Brucker, P. (1988): An Efficient Algorithm for the Job-Shop Problem with Two Jobs. *Computing* 40, 353-359.
- Brucker, P.; A. Drexl, R. Möhring, K. Neumann, and E. Pesch (1999): Resource-Constrained Project Scheduling – Notation, Classification, Models, and Methods. *European Journal of Operational Research* 112, 3-41.
- Brucker, P. and S. Knust (1998): A Linear Programming and Constraint Propagation-Based Lower Bound for the RCPSP. Working Paper, Department of Mathematics, University of Osnabrück.
- Brucker, P. and S. Knust (1999 a): Private Communication.
- Brucker, P. and S. Knust (1999 b): Solving Large-Sized Resource-Constrained Project Scheduling Problems. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 27-51.
- Brucker, P.; S. Knust, A. Schoo, and O. Thiele (1998): A Branch & Bound Algorithm for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research* 107, 272-288.
- Brucker, P.; A. Schoo, and O. Thiele (1996): A Branch & Bound Algorithm for the Resource-Constrained Project Scheduling Problem. Working Paper, Department of Mathematics, University of Osnabrück.
- Buell, C.K. (1967): When to Terminate a Research and Development Project. *Research Management* 10, 275-284.
- Burgess, A.R. and J.B. Killebrew (1962): Variation in Activity Level on a Cyclical Arrow Diagram. *Journal of Industrial Engineering* 13, 76-83.
- Burghardt, M. (1997): *Projektmanagement: Leitfaden für die Planung, Überwachung und Steuerung von Entwicklungsprojekten*. 4th Ed., Publicis MCD, Erlangen.
- Burke, R. (1992): *Project Management – Planning and Control*. 2nd Ed., Wiley, Chichester.

- Carlier, J. and B. Latapie (1991):** Une Méthode Arborescente pour Résoudre les Problèmes Cumulatifs. *RAIRO Recherche Opérationnelle* 25, 311-340.
- Carlton, W.B. and J.W. Barnes (1996):** A Note on Hashing Functions and Tabu Search Algorithms. *European Journal of Operational Research* 95, 237-239.
- Carruthers, J.A. and A. Battersby (1966):** Advances in Critical Path Methods. *Operational Research Quarterly* 17, 359-380.
- Chapman, C. and S. Ward (1997):** Project Risk Management – Processes, Techniques and Insights. Wiley, Chichester.
- Chatzoglou, P.D. and L.A. Macaulay (1996):** A Review of Existing Models for Project Planning and Estimation and the Need for a New Approach. *International Journal of Project Management* 14, 173-183.
- Chen, Y.-L.; D. Rinks, and K. Tang (1997):** Critical Path in an Activity Network with Time Constraints. *European Journal of Operational Research*.
- Cho, J.-H. and Y.-D. Kim (1997):** A Simulated Annealing Algorithm for Resource Constrained Project Scheduling Problems. *Journal of the Operational Research Society* 48, 736-744.
- Christofides, N.; R. Alvarez-Valdés, and J.M. Tamarit (1987):** Project Scheduling with Resource Constraints: A Branch and Bound Approach. *European Journal of Operational Research* 29, 262-273.
- Chun, Y.H.; H. Moskowitz, and R. Plante (1995):** Sequencing a Set of Alternatives under Time Constraints. *Journal of the Operational Research Society* 46, 1133-1144.
- Cleland, D.I. (1999):** Project Management – Strategic Design and Implementation. McGraw-Hill, New York.
- Coffin, M.A. and B.W. Taylor (1996):** Multiple Criteria R&D Project Selection and Scheduling Using Fuzzy Logic. *Computers and Operations Research* 23, 207-220.
- Colorni, A.; M. Dorigo, and V. Maniezzo (1991):** Distributed Optimization by Ant Colonies. In: Varela, F. and P. Bourgine (Eds.): *Proceedings of ECAL-91 – European Conference on Artificial Life*. Elsevier, Amsterdam, 134-142.
- Conway, R.W.; W.L. Maxwell, and L.W. Miller (1967):** Theory of Scheduling. Addison-Wesley, Reading.
- Cook, S.A. (1971):** The Complexity of Theorem-Proving Procedures. *Proceedings of the 3rd Annual Symposium on Theory of Computing*, New York, 151-158.
- Cooper, D.F. (1976):** Heuristics for Scheduling Resource Constrained Projects: An Experimental Investigation. *Management Science* 22, 1186-1194.
- Cooper, D.F. (1977):** A Note on Serial and Parallel Heuristics for Resource Constrained Project Scheduling. *Foundations of Control Engineering* 2, 131-134.
- Cooper, D. and C. Chapman (1987):** Risk Analysis for Large Projects: Models, Methods, and Cases. Wiley, Chichester.
- Cox, M.A.A. (1995):** Simple Normal Approximation to the Completion Time Distribution for a PERT network. *International Journal of Project Management* 13, 265-270.

- Crandall, K. (1973): Project Planning with Precedence Lead-Lag Factors. *Project Management Quarterly* 4, Issue 3, 18-27.
- Dammeyer, F.; P. Forst, and S. Voß (1991): On the Cancellation Sequence Method of Tabu Search. *ORSA Journal on Computing* 3, 262-265.
- Dammeyer, F. and S. Voß (1993): Dynamic Tabu List Management Using the Reverse Elimination Method. *Annals of Operations Research* 41, 31-46.
- Dantzig, G.B. (1967): All Shortest Routes in a Graph. In: Rosenstiehl, V. (Ed.): *Théories des Graphes*. Dunod, Paris, 91-92.
- Davies, E.M. (1973): An Experimental Investigation of Resource Allocation in Multi-activity Projects. *Operations Research Quarterly* 24, 587-591.
- Davis, E.W. (1966): Resource Allocation in Project Network Models – A Survey. *The Journal of Industrial Engineering* 17, 177-188.
- Davis, E.W. (1969): An Exact Algorithm for the Multiple Constrained-Resource Project Scheduling Problem. Unpublished Ph.D. Thesis, Yale University.
- Davis, E.W. (1973): Project Scheduling under Resource Constraints – Historical Review and Categorization of Procedures. *AIIE Transactions* 5, 297-313.
- Davis, E.W. (1975): Project Network Summary Measures Constrained-Resource Scheduling. *AIIE Transactions* 7, 132-142.
- Davis, E.W. and G.E. Heidorn (1971): An Algorithm for Optimal Project Scheduling under Multiple Resource Constraints. *Management Science* 17, B803-B816.
- Davis, E.W. and J.H. Patterson (1975): A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling. *Management Science* 21, 944-955.
- Davis, K.R.; A. Stam, and R. Grzybowski (1992): Resource Constrained Project Scheduling with Multiple Objectives: A Decision Support System. *Computers & Operations Research* 19, 657-669.
- Dawson, C.W. (1995): A Dynamic Sampling Technique for the Simulation of Probabilistic and Generalized Activity Networks. *Omega* 23, 557-566.
- Dayanand, N. and R. Padman (1997): On Modelling Payments in Projects. *Journal of the Operational Research Society* 48, 906-918.
- De, P.; E.J. Dunne, J.B. Ghosh, and C.E. Wells (1995): The Discrete Time-Cost Problem Revisited. *European Journal of Operational Research* 81, 225-238.
- Dean, B.V. and M.J. Nishry (1965): Scoring and Profitability Models for Evaluating and Selecting Engineering Projects. *Operations Research* 13, 550-569.
- Deckro, R.F. and J.E. Hebert (1989): Resource Constrained Project Crashing. *Omega* 17, 69-79.
- Deckro, R.F.; J.E. Hebert, W.A. Verdini, P.H. Grimsrud, and S. Venkateshwar (1995): Nonlinear Time/Cost Tradeoff Models in Project Management. *Computers & Industrial Engineering* 28, 219-229.

- Deckro, R.F.; E.P. Winkovsky, J.E. Hebert, and R. Gagnon (1991): A Decomposition Approach to Multi-Project Scheduling. *European Journal of Operational Research* 51, 110-118.
- De La Garza, J.M. and M.C. Vorster (1991): Total Float Traded as Commodity. *Journal of Construction Engineering and Management* 117, 716-727.
- Della Croce, F. (1995): Generalized Pairwise Interchanges and Machine Scheduling. *European Journal of Operational Research* 83, 310-319.
- Della Croce, F.; R.Tadei, and R. Rolando (1993): Solving a Real World Project Scheduling Problem with a Genetic Approach. *Belgian Journal of Operations Research, Statistics and Computer Science* 33, 65-78.
- Dell'Amico, M. and M. Trubian (1993): Applying Tabu-Search to the Job-Shop Scheduling Problem. *Annals of Operations Research* 41, 231-252.
- Demeulemeester, E.L. (1992): Optimal Algorithms for Various Classes of Multiple Resource-Constrained Project Scheduling Problems. Unpublished Ph.D. Thesis, Department of Applied Economics, Katholieke Universiteit Leuven.
- Demeulemeester, E.L. (1995): Minimizing Resource Availability Costs in Time-Limited Project Networks. *Management Science* 41, 1590-1598.
- Demeulemeester, E.L.; B. Dodin, and W.S. Herroelen (1993): A Random Activity Network Generator. *Operations Research* 41, 972-980.
- Demeulemeester, E.L. and W.S. Herroelen (1992): A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem. *Management Science* 38, 1803-1818.
- Demeulemeester, E.L. and W.S. Herroelen (1996 a): An Efficient Optimal Solution Procedure for the Preemptive Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research* 90, 334-348.
- Demeulemeester, E.L. and W.S. Herroelen (1996 b): Modeling Setup Times, Process Batches and Transfer Batches Using Activity Network Logic. *European Journal of Operational Research* 89, 355-365.
- Demeulemeester, E.L. and W.S. Herroelen (1997 a): A Branch-and-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem. *Operations Research* 45, 201-212.
- Demeulemeester, E.L. and W.S. Herroelen (1997 b): New Benchmark Results for the Resource-Constrained Project Scheduling Problem. *Management Science* 43, 1485-1492.
- Demeulemeester, E.L.; W.S. Herroelen, and S.E. Elmaghraby (1996): Optimal Procedures for the Discrete Time/Cost Trade-Off Problem. *European Journal of Operational Research* 88, 50-68.
- Demeulemeester, E.L.; W.S. Herroelen, W.P. Simpson, S. Baroum, J.H. Patterson, and K.-K. Yang (1994): On a Paper by Christofides et al. for Solving the Multiple Resource-Constrained, Single Project Scheduling Problem. *European Journal of Operational Research* 76, 218-228.

- De Reyck, B. (1995): On the Use of the Restrictiveness as a Measure of Complexity for Resource-Constrained Project Scheduling. Working Paper No. 9535, Department of Applied Economics, Katholieke Universiteit Leuven.
- De Reyck, B. (1998): Scheduling Projects with Generalized Precedence Relations: Exact and Heuristic Procedures. Unpublished Ph.D. Thesis, Department of Applied Economics, Katholieke Universiteit Leuven.
- De Reyck, B.; E.L. Demeulemeester, and W. Herroelen (1999): Algorithms for Scheduling Projects with Generalized Precedence Relations. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 77-106.
- De Reyck, B. and W.S. Herroelen (1996): On the Use of the Complexity Index as a Measure of Complexity in Activity Networks. *European Journal of Operational Research* 91, 347-366.
- De Reyck, B. and W.S. Herroelen (1998 a): A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations. *European Journal of Operational Research* 111, 152-174.
- De Reyck, B. and W.S. Herroelen (1998 b): An Optimal Procedure for the Resource-Constrained Project Scheduling Problem with Discounted Cash Flows and Generalized Precedence Relations. *Computers & Operations Research* 25, 1-17.
- De Werra, D. and A. Hertz (1989): Tabu Search Techniques: A Tutorial and an Application to Neural Networks. *OR Spektrum* 11, 131-141.
- De Wit, J. and W.S. Herroelen (1990): An Evaluation of Microcomputer-Based Software Packages for Project Management. *European Journal of the Operational Research* 49, 102-139.
- Dimsdale, B. (1963): Computer Construction of Minimal Project Networks. *IBM Systems Journal*, March Issue, 24-36.
- Dinsmore, P.C. (1993): A Conceptual Team-Building Model: Achieving Teamwork through Improved Communication and Interpersonal Skills. In: Dinsmore, P.C. (Ed.): *The AMA Handbook of Project Management*. Amacom, New York, 167-176.
- Doersch, R.H. and J.H. Patterson (1977): Scheduling a Project to Maximize its Present Value: A Zero-One Programming Approach. *Management Science* 23, 882-889.
- Domschke, W. (1995): Logistik: Transport. 4th Ed., Oldenbourg, München.
- Domschke, W. (1997): Logistik: Rundreisen und Touren. 4th Ed., Oldenbourg, München.
- Domschke, W. and A. Drexl (1991): Kapazitätsplanung in Netzwerken – Ein Überblick über neuere Modelle und Verfahren. *OR Spektrum* 13, 63-76.
- Domschke, W. and A. Drexl (1996): Logistik: Standorte. 4th Ed., Oldenbourg, München.
- Domschke, W. and A. Drexl (1998): Einführung in Operations Research. 4th Ed., Springer, Berlin.
- Domschke, W.; R. Klein, and A. Scholl (1996 a): Tabu Search – Lösungsstrategie für komplexe Optimierungsprobleme. *WiSt – Wirtschaftswissenschaftliches Studium* 25, 606-610.

- Domschke, W.; R. Klein, and A. Scholl (1996 b): Taktische Tabus – Tabu Search: Durch Verbote schneller optimieren. *c't – Magazin für Computertechnik*, Issue 12, 326-332.
- Domschke, W. and A. Scholl (2000): Grundlagen der Betriebswirtschaftslehre – Eine Einführung aus entscheidungsorientierter Sicht. To appear: Springer, Berlin.
- Domschke, W.; A. Scholl, and S. Voß (1997): Produktionsplanung – Ablauforganisatorische Aspekte. 2nd Ed., Springer, Berlin.
- Dorndorf, U.; E. Pesch, and T. Phan Huy (1998): A Time-Oriented Branch-and-Bound Algorithm for Resource-Constrained Project Scheduling with Generalised Precedence Constraints. Working Paper, Faculty of Economics, University of Bonn.
- Dorndorf, U.; T. Phan Huy, and E. Pesch (1999): A Survey of Interval Capacity Consistency Tests for Time- and Resource-Constrained Scheduling. To appear in: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 213-238.
- Dorigo, M.; V. Maniezzo, and A. Colorni (1996): Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics* B-26, 29-41.
- Dowsland, K. (1993): Simulated Annealing. In: Reeves, C. (Ed.): *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell, Oxford, 20-69.
- Dowsland, K. (1996): Genetic Algorithms – A Tool for OR? *Journal of the Operational Research Society* 47, 550-561.
- Drexel, A. (1988): A Simulated Annealing Approach to the Multi-Constraint Zero-One Knapsack Problem. *Computing* 40, 1-8.
- Drexel, A. (1990): Fließbandaustaktung, Maschinenbelegung und Kapazitätsplanung in Netzwerken – Ein integrierender Ansatz. *Zeitschrift für Betriebswirtschaftslehre* 60, 53-70.
- Drexel, A. (1991): Scheduling of Project Networks by Job Assignment. *Management Science* 37, 1590-1602.
- Drexel, A.; W. Eversheim, H. Esser, and R. Grempe (1994 a): CIM im Werkzeugmaschinenbau: Der Prisma-Montageleitstand. *Zeitschrift für betriebswirtschaftliche Forschung* 46, 279-295.
- Drexel, A.; B. Fleischmann, H.O. Günther, H. Stadtler, and H. Tempelmeier (1994 b): Konzeptionelle Grundlagen kapazitätsorientierter PPS-Systeme. *Zeitschrift für betriebswirtschaftliche Forschung* 46, 1022-1045.
- Drexel, A. and J. Grünewald (1993): Nonpreemptive Multi-Mode Resource-Constrained Project Scheduling. *IIE Transactions* 25, 74-81.
- Drexel, A.; J. Juretzka, F. Salewski, and A. Schirmer (1999): New Modelling Concepts and Their Impact on Resource-Constrained Project Scheduling. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 413-432.
- Drexel, A. and A. Kimms (1998 a): Minimizing Total Weighted Completion Times Subject to Precedence Constraints by Dynamic Programming. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 475, University of Kiel.

- Drexl, A. and A. Kimms (1998 b): Optimization Guided Lower and Upper Bounds for the Resource Investment Problem. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 481, University of Kiel.
- Drexl, A. and R. Kolisch (1993 a): Produktionsplanung und -steuerung bei Einzel- und Kleinserienfertigung – Grundlagen. *WiSt – Wirtschaftswissenschaftliches Studium* 22, 60-66.
- Drexl, A. and R. Kolisch (1993 b): Produktionsplanung und -steuerung bei Einzel- und Kleinserienfertigung – Leitstandskonzepte. *WiSt – Wirtschaftswissenschaftliches Studium* 22, 137-141.
- Drexl, A. and R. Kolisch (1996): Assembly Management in Machine Tool Manufacturing and the PRISMA-Leitstand. *Production and Inventory Management Journal* 37, Issue 4, 55-57.
- Drexl, A.; R. Kolisch, and A. Sprecher (1997): Neuere Entwicklungen in der Projektplanung. *Zeitschrift für betriebswirtschaftliche Forschung* 49, 95-120.
- Drexl, A.; R. Kolisch, and A. Sprecher (1998): Koordination und Integration im Projektmanagement – Aufgaben und Instrumente. *Zeitschrift für Betriebswirtschaft* 68, 275-295.
- Dumond, J. (1992): In a Multi-Resource Environment – How Much Is Enough? *International Journal of Production Research* 30, 395-410.
- Dumond, J. and V.A. Mabert (1988): Evaluating Project Scheduling and Due Date Assignment Procedures: An Experimental Analysis. *Management Science* 34, 101-118.
- Dworatschek, S. and A. Hayek (1992): *Marktspiegel Projektmanagementsoftware: Kriterienkatalog und Leistungsprofile*. 3rd Ed., Verlag TÜV Rheinland, Köln.
- Easa, S.M. (1989): Resource Leveling in Construction by Optimization. *Journal of Construction Engineering and Management* 115, 302-316.
- Eggerman, W.V. (1990): *Configuration Management Handbook*. TAB Books, Blue Ridge Summit.
- Ellis, R.D. (1993): Project Cost Control Systems That Really Work. In: Dinsmore, P.C. (Ed.): *The AMA Handbook of Project Management*. Amacom, New York, 167-176.
- Elmaghraby, S.E. (1977): *Activity Networks: Project Planning and Control by Network Models*. Wiley, Chichester.
- Elmaghraby, S.E. (1993): Resource Allocation via Dynamic Programming in Activity Networks. *European Journal of Operational Research* 64, 199-215.
- Elmaghraby, S.E. (1995): Activity Nets: A Guided Tour through Some Recent Developments. *European Journal of Operational Research* 82, 383-408.
- Elmaghraby, S.E. and W.S. Herroelen (1980): On the Measurement of Complexity in Activity Networks. *European Journal of Operational Research* 5, 223-234.
- Elmaghraby, S.E. and W.S. Herroelen (1990): The Scheduling of Activities to Maximize the Net Present Value of Projects. *European Journal of Operational Research* 49, 35-49.

- Elmaghraby, S.E. and J. Kamburowsky (1990): On Project Representations and Activity Floats. *The Arabian Journal for Science and Engineering* 14, 627-637.
- Elmaghraby, S.E. and J. Kamburowsky (1992): The Analysis of Activity Networks under Generalized Precedence Constraints. *Management Science* 38, 1245-1263.
- Elsayed, E.A. (1982): Algorithms for Project Scheduling with Resource Constraints. *International Journal of Production Research* 20, 95-103.
- Elsayed, E.A. and N.Z. Nasr (1986): Heuristics for Resource-Constrained Project Scheduling. *International Journal of Production Research* 24, 299-310.
- Etgar, R.; A. Shtub, and L.J. LeBlanc (1996): Scheduling Projects to Maximize the Net Present Value – The Case of Time-Dependent, Contingent Cash Flows. *European Journal of Operational Research* 96, 90-96.
- Fadlalla, A. and J.R. Evans (1995): Improving the Performance of Enumerative Search Methods: I. Exploiting Structure and Intelligence. *Computers & Operations Research* 22, 605-613.
- Farid, F. and S. Manoharan (1996): Comparative Analysis of Resource-Allocation Capabilities of Project Management Software Packages. *Project Management Journal* 26, Issue 2, 35-44.
- Fehler, D.W. (1969): Die Variationen-Enumeration – ein Näherungsverfahren zur Planung des optimalen Betriebsmitteleinsatzes bei der Terminierung von Projekten. *Elektronische Datenverarbeitung* 10, 479-483.
- Fink, A.; G. Schneidereit, and S. Voß (1998): Ring Network Design for Metropolitan Area Networks. Working Paper, Institut für Wirtschaftswissenschaften, Technical University of Braunschweig.
- Fink, A. and S. Voß (1998): Applications of Modern Heuristic Search Methods to Continuous Flow-Shop Sequencing Problems. Working Paper, Institut für Wirtschaftswissenschaften, Technical University of Braunschweig.
- Fink, A. and S. Voß (1999): Applications of Modern Heuristic Search Methods to Pattern Sequencing Problems. *Computers & Operations Research* 26, 17-34.
- Fischetti, M. and P. Toth (1989): An Additive Bounding Procedure for Combinatorial Optimization Problems. *Operations Research* 37, 319-328.
- Fisher, M.L. (1973): Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I. *Operations Research* 21, 1114-1127.
- Fisher, M.L. (1981): The Lagrangean Relaxation Method for Solving Integer Programming Problems. *Management Science* 27, 1-18.
- Floyd, R. (1962): Algorithm 97: Shortest Path. *Communications of the ACM* 5, 345.
- Fox, G.E.; N.R. Baker, and J.L. Bryant (1984): Economic Models for R&D Project-Selection in the Presence of Project Interactions. *Management Science* 30, 890-902.
- Franck, B. and K. Neumann (1997): Resource-Constrained Project Scheduling with Time Windows – Structural Questions and Priority-Rule Methods. Technical Report WIOR 512, University of Karlsruhe.

- Franck, B.; K. Neumann, and Ch. Schwindt (1997): A Capacity-Oriented Hierarchical Approach to Single-Item and Small-Batch Production Planning Using Project Scheduling Methods. *OR Spektrum* 19, 77-85.
- Franck, B. and Th. Selle (1998): Metaheuristics for the Resource-Constrained Project Scheduling with Schedule-Dependent Time Windows. Technical Report WIOR 546, University of Karlsruhe.
- French, S. (1982): Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop. Horwood, Chichester.
- Fulkerson, D.R. (1961): A Network Flow Computation for Project Cost Curves. *Management Science* 7, 167-178.
- Garey, M.R. and D.S. Johnson (1979): Computers and Intractability – A Guide to the Theory of NP-Completeness. Freeman, New York.
- Garey, M.R.; D.S. Johnson, and R. Sethi (1976): The Complexity of Flowshop and Job-shop Scheduling. *Mathematics of Operations Research* 1, 117-129.
- Gavish, B. and H. Pirkul (1985): Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality. *Mathematical Programming* 31, 78-105.
- Gear, T.E. and G.C. Cowie (1980): A Note on Modelling Project Interdependence in Research Development. *Decision Sciences* 11, 738-748.
- Gendron, B. and T.G. Crainic (1994): Parallel Branch-and-Bound Algorithms: Survey and Synthesis. *Operations Research* 42, 1042-1067.
- Geoffrion, A.M. (1974): Lagrangean Relaxation for Integer Programming. *Mathematical Programming Study* 2, 82-114.
- Geoffrion, A.M. and R.E. Marsten (1972): Integer Programming Algorithms: A Framework and State-of-the-Art Survey. *Management Science* 18, 465-491.
- Glover, F. (1986): Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research* 5, 533-549.
- Glover, F. (1989): Tabu Search – Part I. *ORSA Journal on Computing* 1, 190-206.
- Glover, F. (1990): Tabu Search – Part II. *ORSA Journal on Computing* 2, 4-32.
- Glover, F. and H.J. Greenberg (1989): New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence. *European Journal of Operational Research* 39, 119-130.
- Glover, F. and M. Laguna (1997): Tabu Search. Kluwer, Boston.
- Goldberg, D.E. (1989): Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley, Reading.
- Gonguet, L. (1969): Comparison of Three Heuristic Procedures for Allocating Resources and Producing Schedules. In: Lombaers, H.J.M. (Ed.): *Project Planning by Network Analysis*. North-Holland, Amsterdam, 249-255.
- Grinold, R.C. (1972): The Payment Scheduling Problem. *Naval Research Logistics Quarterly* 19, 123-136.

- Gulledge, T.R.; W.P. Hutzler, and J.S. Lovelace (1992): Cost Estimating and Analysis – Balancing Technology and Declining Budgets. Springer, Berlin.
- Güven Iyigün, M. (1993): A Decision Support System for R&D Project Selection and Resource Allocation under Uncertainty. *Project Management Journal* 24, Issue 4, 5-13.
- Hall, T.J. (1995): The Quality Systems Manual – The Definite Guide to the ISO 9000 Family and TickIT. Wiley, Chichester.
- Hamacher, H.W. and S. Tufekci (1984): Algebraic Flows and Time-Cost Tradeoff Problems. *Annals of Discrete Mathematics* 19, 165-182.
- Hand, M. and B. Plowman (Eds.) (1992): Quality Management Handbook. Butterworth-Heinemann, Oxford.
- Hapke, M.; A. Jaszkiewicz, and R. Slowinski (1999): Fuzzy Multi-Mode Resource-Constrained Project Scheduling with Multiple Objectives. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 355-382.
- Harhalakis, G. (1989): Evaluation of Resource Allocations in Project Scheduling. In: Slowinski, R. and J. Weglarz (Eds.): *Advances in Project Scheduling*. Elsevier, Amsterdam, 67-86.
- Harris, R.B. (1990): Packing Method for Resource Leveling (PACK). *Journal of Construction Engineering and Management* 116, 331-350.
- Hartmann, S. (1998): A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling. *Naval Research Logistics* 45, 733-750.
- Hartmann, S. (1997 a): Project Scheduling with Multiple Modes: A Genetic Algorithm. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 435, University of Kiel.
- Hartmann, S. (1997 b): Scheduling Medical Research Experiments – An Application of Project Scheduling Methods. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 452, University of Kiel.
- Hartmann, S. and A. Drexl (1998): Project Scheduling with Multiple Modes: A Comparison of Exact Algorithms. *Networks* 32, 238-257.
- Hartmann, S. and R. Kolisch (1998): Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem. To appear in: *European Journal of Operational Research*.
- Hartmann, S. and A. Sprecher (1996): A Note on "Hierarchical Models for Multi-Project Planning and Scheduling". *European Journal of Operational Research* 94, 377-383.
- Harvey, R.T. and J.H. Patterson (1997): An Implicit Enumeration Algorithm for the Time/Cost Tradeoff Problem. *Foundation of Control Engineering* 4, 107-117.
- Hastings, N.A.J. (1972): On Resource Allocation in Project Networks. *Operational Research Quarterly* 23, 217-221.
- Heilmann, R. (1998): A Branch-and-Bound Procedure for MRCPSP/max. Technical Report WIOR 512, University of Karlsruhe.

- Heilmann, R. and Ch. Schwindt (1997): Lower Bounds for RCPSP/max. Technical Report WIOR 511, University of Karlsruhe.
- Heindel, L.E. and V.A. Kasten (1996): Next-Generation PC-Based Project Management Systems: The Path Forward. *International Journal of Project Management* 14, 249-253.
- Helbrough, B. (1995): Computer Assisted Collaboration – The Fourth Dimension of Project Management. *International Journal of Project Management* 13, 329-333.
- Held, M.; P. Wolfe, and H.P. Crowder (1974): Validation of Subgradient Optimization. *Mathematical Programming* 6, 62-68.
- Herbst, A.F. (1990): *The Handbook of Capital Investing – Analysis and Strategies for Investment in Capital Assets*. Harper Business, New York.
- Herroelen, W.S. (1972): Resource-Constrained Project Scheduling – The State of the Art. *Operational Research Quarterly* 23, 261-275.
- Herroelen, W.S.; E.L. Demeulemeester, and B. de Reyck (1998): Resource-Constrained Project Scheduling – A Survey of Recent Developments. *Computers & Operations Research* 25, 279-302.
- Herroelen, W.S.; E.L. Demeulemeester, and B. de Reyck (1999): A Classification Scheme for Project Scheduling. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 1-26.
- Herroelen, W.S. and E. Gallens (1993): Computational Experience with an Optimal Procedure for the Scheduling of Activities to Maximize the Net Present Value. *European Journal of Operational Research* 65, 274-277.
- Herroelen, W.S.; P. Van Dommelen, and E.L. Demeulemeester (1997): Project Network Models with Discounted Cash Flows – A Guided Tour through Recent Developments. *European Journal of Operational Research* 100, 97-121.
- Hess, S.W. (1993): Swinging on the Branch of Tree: Project Selection Applications. *Interfaces* 23, 5-12.
- Hillier, F.S. and G.J. Lieberman (1995): *Introduction to Operations Research*. 6th Ed., McGraw-Hill, New York.
- Hobbs, B. and P. Ménard (1993): Organizational Choices for Project Management. In: Dinsmore, P.C. (Ed.): *The AMA Handbook of Project Management*. Amacom, New York, 81-108.
- Holland, J.H. (1975): *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.
- Hopfield, J.J. and D. Tank (1985): Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics* 52, 141-152.
- Householder, J. and H.E. Rutland (1990): Who Owns Float? *Journal of Construction Engineering and Management* 116, 130-133.
- Hubbard, D.G. (1993): Work Structuring. In: Dinsmore, P.C. (Ed.): *The AMA Handbook of Project Management*. Amacom, New York, 71-78.

- Hutchison, J. and Y.-L. Chang (1990): Optimal Nondelay Job Shop Schedules. International Journal of Production Research 28, 245-257.
- Ibaraki, T. (1977): The Power of Dominance Relations in Branch-and-Bound Algorithms. Journal of the Association for Computing Machinery 24, 264-279.
- Icmeli, O. and W.O. Rom (1996): Solving the Resource-Constrained Project Scheduling Problem with the Optimization Subroutine Library. Computers & Operations Research 23, 801-817.
- Icmeli, O. and S. Selcuk Erenguc (1994): A Tabu Search Procedure for the Resource Constrained Project Scheduling Problem with Discounted Cash Flows. Computers & Operations Research 8, 841-853.
- Icmeli, O. and S. Selcuk Erenguc (1996): A Branch and Bound Procedure for the Resource Constrained Project Scheduling Problem with Discounted Cash Flows. Management Science 42, 1395-1408.
- Icmeli, O.; S. Selcuk Erenguc, and Ch.J. Zappe (1993): Project Scheduling Problems: A Survey. International Journal of Operations & Production Management 13, 80-91.
- Icmeli-Tukel, O. and W.O. Rom (1998): Analysis of the Characteristics of Projects in Diverse Industries. Journal of Operations Management 16, 43-61.
- Icmeli-Tukel, O. and W.O. Rom (1997): Ensuring Quality in Resource Constrained Project Scheduling. European Journal of Operational Research 103, 483-496.
- Jackson, J.R. (1956): A Computing Procedure for a Line Balancing Problem. Management Science 2, 261-271.
- Johnson, R.V. (1988): Efficient Modular Implementation of Branch-and-Bound Algorithms. Decision Sciences 19, 17-38.
- Johnson, R.V. (1992): Resource Constrained Project Scheduling Capabilities of Commercial Project Management Software. Project Management Journal 22, Issue 4, 39-43.
- Johnson, T.J.R. (1967): An Algorithm for the Resource-Constrained Project Scheduling Problem. Unpublished Ph.D. Thesis, Massachusetts Institute of Technology.
- Jüngen, F.J. and W. Kowalczyk (1995): An Intelligent Interactive Project Management Support System. European Journal of Operational Research 84, 60-81.
- Just, M.R. and J.P. Murphy (1994): The Effect of Resource Constraints on Project Schedules. AACE Transactions, DCL.2.1-DCL.2.6.
- Kaimann, R.A. (1974): Coefficient of Network Complexity. Management Science 21, 172-177.
- Kallo, G.G. (1996): The Reliability of Critical Path Method (CPM) Techniques in Analysis and Evaluation of Delay Claims. Cost Engineering 38, Issue 5, 35-37.
- Kamburowski, J. (1997): New Validations of PERT Times. Omega 25, 323-328.
- Kaplan, L.A. (1988): Resource-Constrained Project Scheduling with Preemption of Jobs. Unpublished Ph.D. Thesis, University of Michigan.

- Karp, R.M. (1972): Reducibility Among Combinatorial Problems. In: Miller, R.E. and J.W. Thatcher (Eds.): Complexity of Computer Computation. Plenum Press, New York, 85-103.
- Katzaz, B. and C. Sepil (1996): Project Scheduling with Discounted Cash Flows and Progress Payments. *Journal of the Operational Research Society* 47, 1262-1272.
- Kelley, J.E. (1961): Critical-Path Planning and Scheduling: Mathematical Basis. *Operations Research* 9, 296-320.
- Kelley, J.E. (1963): The Critical-Path Method: Resources Planning and Scheduling. In: Muth, J.F. and G.L. Thompson (Eds.): Industrial Scheduling. Prentice Hall, Englewood Cliffs, 347-365.
- Keown, A.L.; D.F. Scott, J.D. Martin, and J.W. Petty (1994): Foundations of Finance – The Logic and Practice of Financial Management. Prentice Hall, Englewood Cliffs.
- Kerzner, H. (1998): Project Management – A Systems Approach to Planning, Scheduling, and Controlling. Van Nostrand Reinhold, New York.
- Khattab, M.M. and F. Choobineh (1991): A New Approach for Project Scheduling with a Limited Resource. *International Journal of Production Research* 29, 185-198.
- Khattab, M.M. and K. Soyland (1996): Limited-Resource Allocation in Construction Projects. *Computers and Industrial Engineering* 31, 229-232.
- Kim, S.Y. and R.C. Leachman (1993): Multi-Project Scheduling with Explicit Lateness Costs. *IIE Transactions* 25, 34-44.
- Kim, S.O. and M.J. Schniederjans (1989): Heuristic Framework for the Resource-Constrained Multi-Project Scheduling Problem. *Computers & Operations Research* 16, 541-556.
- Kimms, A. (1998): Fallbasiertes Schließen auf Methoden zur Produktionsplanung. *Wirtschaftsinformatik* 40, 417-423.
- Kirkpatrick, S.; C.D. Gelatt, and M.P. Vecchi (1983): Optimization by Simulated Annealing. *Science* 220, 671-680.
- Klein, R. (1998): Bidirectional Planning – Improving Priority Rule Based Heuristics for Scheduling Resource-Constrained Projects. To appear in: *European Journal of Operational Research*.
- Klein, R. and A. Scholl (1996): Maximizing the Production Rate in Simple Assembly Line Balancing – A Branch and Bound Procedure. *European Journal of Operational Research* 91, 367-385.
- Klein, R. and A. Scholl (1998 a): PROGRESS: Optimally Solving the Generalized Resource-Constrained Project Scheduling Problem. *Schriften zur Quantitativen Betriebswirtschaftslehre* 5/98, Darmstadt University of Technology.
- Klein, R. and A. Scholl (1998 b): Scattered Branch and Bound – An Adaptive Search Strategy Applied to Resource-Constrained Project Scheduling. *Schriften zur Quantitativen Betriebswirtschaftslehre* 6/98, Darmstadt University of Technology.
- Klein, R. and A. Scholl (1999): Computing Lower Bounds by Destructive Improvement: An Application to Resource-Constrained Project Scheduling. *European Journal of Operational Research* 112, 322-346.

- Knolmayer, G. and D. Rückle (1976): Betriebswirtschaftliche Grundlagen der Projekt kostenminimierung in Projektnetzwerken. Zeitschrift für betriebswirtschaftliche Forschung 28, 431-447.
- Kohler, W.H. and K. Steiglitz (1974): Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems. Journal of the Association for Computing Machinery 21, 140-156.
- Kohlmorgen, U.; H. Schmeck, and K. Haase (1996): Experiences with Fine-Grained Parallel Genetic Algorithms. To appear in: Annals of OR.
- Kolen, A. and E. Pesch (1994): Genetic Local Search in Combinatorial Optimization. Discrete Applied Mathematics 48, 273-284.
- Kolisch, R. (1995): Project Scheduling under Resource Constraints – Efficient Heuristics for Several Problem Classes. Physica, Heidelberg.
- Kolisch, R. (1996 a): Efficient Priority Rules for the Resource-Constrained Project Scheduling Problem. Journal of Operations Management 14, 179-192.
- Kolisch, R. (1996 b): Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation. European Journal of Operational Research 90, 320-333.
- Kolisch, R. (1997): Investitionsplanung in Netzwerken. Zeitschrift für Betriebswirtschaftslehre 67, 1057-1072.
- Kolisch, R. and A. Drexl (1996): Adaptive Search for Solving Hard Project Scheduling Problems. Naval Research Logistics 43, 23-40.
- Kolisch, R. and A. Drexl (1997): Local Search for Nonpreemptive Multi-Mode Resource-Constrained Project Scheduling. IIE Transactions 29, 987-999.
- Kolisch, R. and S. Hartmann (1999): Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. In: Weglarz, J. (Ed.): Handbook on Recent Advances in Project Scheduling. Kluwer, Dordrecht, 147-178.
- Kolisch, R. and K. Hempel (1996 a): Auswahl von Standardsoftware, dargestellt am Beispiel von Programmen für das Projektmanagement. Wirtschaftsinformatik 38, 399-410.
- Kolisch, R. and K. Hempel (1996 b): Experimentelle Evaluation der Kapazitätsplanung von Projektmanagementsoftware. Zeitschrift für betriebswirtschaftliche Forschung 48, 999-1018.
- Kolisch, R. and K. Hempel (1996 c): Finite Scheduling Capabilities of Commercial Project Management Systems. Manuskripte aus den Instituten für Betriebswirtschaftslehre 397, University of Kiel.
- Kolisch, R. and A. Sprecher (1996): PSPLIB - A Project Scheduling Problem Library. European Journal of Operational Research 96, 205-216.
- Kolisch, R. and R. Padman (1997): An Integrated Survey of Project Scheduling – Models, Algorithms, Problems, and Applications. Manuskripte aus den Instituten für Betriebswirtschaftslehre 463, University of Kiel.

- Kolisch, R.; A. Sprecher, and A. Drexl (1995): Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science* 41, 1693-1703.
- Kolisch, R.; Ch. Schwindt, and A. Sprecher (1999): Benchmark Instances for Project Scheduling Problems. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 197-212.
- Krishnamoorthy, M.S. and N. Deo (1979): Complexity of the Minimum-Dummy-Activities Problem in a PERT Network. *Networks* 9, 189-194.
- Küpper, W.; K. Lüder, and L. Streitferdt (1975): *Netzplantechnik*. Physica, Würzburg.
- Kurtulus, I.S. and S.C. Narula (1985): Multi-Project Scheduling: Analysis of Project Performance. *IIE Transactions* 17, 58-65.
- Labbé, M.; G. Laporte, and H. Mercure (1991): Capacitated Vehicle Routing on Trees. *Operations Research* 39, 616-622.
- Lambert, L.R. (1993): Cost/Schedule Control System Criteria (C/SCSC): An Integrated Project Management Approach Using Earned Value Analysis. In: Dinsmore, P.C. (Ed.): *The AMA Handbook of Project Management*. Amacom, New York, 167-176.
- Lau, H.-S. and A.H.-L. Lau (1998): An Improved PERT-Type Formula for Standard Deviations. *IIE Transactions* 30, 273-275.
- Lau, H.-S. and C. Somarajan (1995): A Proposal on Improved Procedures for Estimating Task-Time Distributions in PERT. *European Journal of Operational Research* 85, 39-52.
- Laue, H.J. (1968): Efficient Methods for the Allocation of Resources in Project Networks. *Unternehmensforschung* 12, 133-143.
- Lawrence, S.R. and Th.E. Morton (1993): Resource-Constrained Multi-Project Scheduling with Tardy Costs: Comparing Myopic, Bottleneck, and Resource Pricing Heuristics. *European Journal of Operational Research* 64, 168-187.
- Leachman, R.C. and S. Kim (1993): A Revised Critical Path Method for Networks Including Both Overlap Relationships and Variable-Duration Activities. *European Journal of Operational Research* 64, 229-248.
- Lee, J.-K. and Y.-D. Kim (1996): Search Heuristics for Resource Constrained Project Scheduling. *Journal of the Operational Research Society* 47, 678-689.
- Lee-Kwang, H. and J. Favrel (1988): The SSD Graph: A Tool for Project Scheduling and Visualization. *IEEE Transactions on Engineering Management* 35, 25-30.
- Leon, V.J. and R. Balakrishnan (1995): Strength and Adaptability of Problem-Space Based Neighborhoods for Resource-Constrained Scheduling. *OR Spektrum* 17, 173-182.
- Levary, R.R. and N.E. Seitz (1990): Quantitative Methods for Capital Budgeting. South-Western Publishing, Cincinnati.
- Levner, E.V. and A.S. Nemirovski (1994): A Network Flow Algorithm for Just-in-Time Project Scheduling. *European Journal of Operational Research* 79, 167-175.

- Levy, F.K.; G.L.Thompson, and J.D. Wiest (1962): Multiship, Multishop, Workload-Smoothing Program. *Naval Research Logistics Quarterly*, 37-44.
- Lewis, J.P. (1997): *Fundamentals of Project Management*. Amacom, New York.
- Lewis, J.P. (1995): *Project Planning, Scheduling, and Control – A Hands-On Guide to Bringing Projects in on Time and on Budget*. Irwin, Chicago.
- Li, R.K.-Y. and R.J. Willis (1991): Alternate Resources in Project Scheduling. *Computers & Operations Research* 8, 663-668.
- Li, R.K.-Y. and R.J. Willis (1992): An Iterative Scheduling Technique for Resource-Constrained Project Scheduling. *European Journal of Operational Research* 56, 370-379.
- Li, R.K.-Y. and R.J. Willis (1993): Resource Constrained Scheduling within Fixed Project Durations. *Journal of the Operational Research Society* 44, 71-80.
- Liberatore, M.J. and G.J. Titus (1983): The Practice of Management Science in R&D Management. *Management Science* 29, 962-974.
- Lock, D. (1984): *Project Management*. Gower, Aldershot.
- Love, S.F. (1983): Save Time and Money on Projects by Using Float. *Project Management Quarterly* 14, Issue 4, 46-49.
- MacLeod, K.R. and P.F. Petersen (1996): Estimating the Tradeoff Between Resource Allocation and Probability of On-Time Completion in Project Management. *Project Management Journal*, Issue 3, 26-33.
- Maniezzo, V. and A. Mingozzi (1999): A Heuristic Procedure for the Multi-Mode Project Scheduling Problem Based on Benders' Decomposition. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 179-196.
- Maroto, C. and P. Tormos (1994): Project Management: An Evaluation of Software Quality. *International Transactions on Operational Research* 1, 209-221.
- Maroto, C.; P. Tormos, and A. Lova (1999): The Evolution of Software Quality in Project Scheduling. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 239-259.
- Martello, S. and P. Toth (1990): Lower Bounds and Reduction Procedures for the Bin Packing Problem. *Discrete Applied Mathematics* 28, 59-70.
- Martino, J.P. (1983): *Technological Forecasting for Decision Making*. North Holland, New York.
- Masson, E. and Y.-J. Wang (1990): Introduction to Computation and Learning in Artificial Neural Networks. *European Journal of Operational Research* 47, 1-28.
- Matthes, W. (1980): Planpufferzeiten in Projektnetzen. *Zeitschrift für Betriebswirtschaft* 50, 127-147.
- McKnew, M.A.; C. Saydam, and B.J. Coleman (1991): An Efficient Zero-One Formulation of the Multi-Level Lotsizing Problem. *Decision Sciences* 22, 280-295.
- Mendelsohn, A.S. (1993): The Essence of Quality Management. In: Dinsmore, P.C. (Ed.): *The AMA Handbook of Project Management*. Amacom, New York, 261-269.

- Meredith, J.R. and S.J. Mantel (1995): Project Management: A Managerial Approach. 3rd Ed., Wiley, Chichester.
- Michael, D.J.; J. Kamburowski, and M. Stallmann (1993): On the Minimum Dummy-Arc Problem. RAIRO Recherche Opérationnelle 27, 153-168.
- Michalewicz, Z. (1994): Genetic Algorithms + Data Structures = Evolution Programs. 2nd Ed., Springer, Berlin.
- Mingozzi, A.; V. Maniezzo, S. Ricciardelli, and L. Bianco (1998): An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. Management Science 44, 715-729.
- Mocellin, J.V. (1989): A Two-Stage Approach to Resource-Constrained Multi-Project Scheduling Problems. Policy and Information 13, 1-18.
- Moder, J.J.; C.R. Phillips, and E.W. Davis (1983): Project Management with CPM, PERT, and Precedence Diagramming. 3rd Ed., Van Nostrand Reinhold, New York.
- Mohanty, R.P. and M.K. Siddiq (1989 a): Multiple Projects – Multiple Resource-Constrained Scheduling: A Multi-Objective Analysis. Engineering Costs and Production Economics 18, 83-92.
- Mohanty, R.P. and M.K. Siddiq (1989 b): Multiple Projects – Multiple Resource-Constrained Scheduling: Some Studies. International Journal of Production Research 27, 261-280.
- Möhring, R.H. (1984): Minimizing Costs of Resource Requirements in Project Networks Subject to a Fixed Completion Time. Operations Research 32, 89-120.
- Möhring, R.H.; F. Stork, and M. Uetz (1998): Resource Constrained Project Scheduling with Time Windows: A Branching Scheme Based on Dynamic Release Dates. Technical Report 596, Department of Mathematics, Technical University of Berlin.
- Moodie, C.L. and D.E. Mandeville (1966): Project Resource Balancing by Assembly Line Balancing Techniques. The Journal of Industrial Engineering 17, 377-383.
- Mori, M. and C.C. Tseng (1997): A Genetic Algorithm for Multi-Mode Resource Constrained Project Scheduling. European Journal of Operational Research 100, 134-141.
- Müller-Merbach, H. (1967 a): Ein Verfahren zur Planung des optimalen Betriebsmitteleinsatzes bei der Terminierung von Großprojekten – 1. Teil. AWF Mitteilungen 45, 83-88.
- Müller-Merbach, H. (1967 b): Ein Verfahren zur Planung des optimalen Betriebsmitteleinsatzes bei der Terminierung von Großprojekten – 2. Teil. AWF Mitteilungen 45, 135-140.
- Munns, A.K. and B.F. Bjeirmi (1996): The Role of Project Management in Achieving Project Success. International Journal of Project Management 14, 81-87.
- Nabrzyski, J. and J. Weglarz (1999): Knowledge-Based Multiobjective Project Scheduling. In: Weglarz, J. (Ed.): Handbook on Recent Advances in Project Scheduling. Kluwer, Dordrecht, 383-412.

- Nair, K.P.K.; V.R. Prasad, and Y.P. Aneja (1993): Efficient Chains in a Network with a Time-Cost Trade-Off Function on Each Arc. *European Journal of Operational Research* 66, 392-402.
- Naphade, K.S.; S.D. Wu, and R.H. Storer (1997): Problem Space Search Algorithms for Resource-Constrained Project Scheduling Problems. *Annals of Operations Research* 70, 307-326.
- Nasution, S.H. (1994): Fuzzy Critical Path Method. *IEEE Transactions on Systems, Manufacturing, and Cybernetics* 24, 48-57.
- Nazareth, T.; S. Verma, S. Bhattacharya, and A. Bagchi (1999): The Multiple Resource Constrained Project Scheduling Problem: A Breadth-First Approach. *European Journal of Operational Research* 112, 347-366.
- Nemhauser, G.L. and L.A. Wolsey (1988): Integer and Combinatorial Optimization. Wiley, Chichester.
- Neumann, K. (1990): Stochastic Project Networks – Temporal Analysis, Scheduling and Cost Minimization. Springer, Berlin.
- Neumann, K. and M. Morlock (1993): Operations Research. Carl Hanser, München.
- Neumann, K.; H. Nübel und Ch. Schwindt (1999): Active and Stable Project Scheduling. Technical Report WIOR 556, University of Karlsruhe.
- Neumann, K. and Ch. Schwindt (1997): Activity-on-Node Networks with Minimal and Maximal Time Lags and their Application to Make-to-Order Production. OR Spektrum 19, 205-217.
- Neumann, K. and J. Zhan (1995): Heuristics for the Minimum Project-Duration Problem with Minimal and Maximal Time-Lags under Fixed Resources. *Journal of Intelligent Manufacturing* 6, 145-154.
- Neumann, K. and J. Zimmermann (1997): Resource Levelling for Projects with Schedule-Dependent Time Windows. Technical Report WIOR 508, University of Karlsruhe.
- Neumann, J. and J. Zimmermann (1998): Exact and Heuristic Procedures for Net Present Value and Resource Levelling in Project Scheduling. Technical Report WIOR 538, University of Karlsruhe.
- Neumann, K. and J. Zimmermann (1999): Methods for Resource-Constrained Project Scheduling with Regular and Non-Regular Objective Functions and Schedule-Dependent Time Windows. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 261-288.
- Nkasu, M.M. and K.H. Leung (1997): A Resources Scheduling Decision Support System for Concurrent Project Management. *International Journal of Production Research* 35, 3107-3132.
- Norbis, M.I. and J.M. Smith (1986): Two Level Heuristic for the Resource Constrained Scheduling Problem. *International Journal of Production Research* 24, 1203-1219.
- Norbis, M. and J.M. Smith (1996): An Interactive Decision Support System for the Resource-Constrained Scheduling Problem. *European Journal of Operational Research* 94, 54-65.

- Nowicki, E. and C. Smutnicki (1994): A Decision Support System for the Resource Constrained Project Scheduling Problem. *European Journal of Operational Research* 79, 183-195.
- Nübel, R. (1998): A Branch-and-Bound Procedure for the Resource Investment Problem with Generalized Precedence Constraints. Technical Report WIOR 516, University of Karlsruhe.
- Nübel, R. and Ch. Schwindt (1997): A Classification of Shifts, Schedules, and Objective Functions in Project Scheduling. Technical Report WIOR 509, University of Karlsruhe.
- Nuijten, W.P. and C. Le Pape (1998): Constraint-Based Job Shop Scheduling with Ilog Scheduler. *Journal of Heuristics* 3, 271-286.
- Oguz, O. and H. Bala (1994): A Comparative Study of Computational Procedures for the Resource Constrained Project Scheduling Problem. *European Journal of Operational Research* 72, 406-416.
- Özdamar, L. and G. Ulusoy (1994): A Local Constraint Based Analysis Approach to Project Scheduling under General Resource Constraints. *European Journal of Operational Research* 79, 287-298.
- Özdamar, L. and G. Ulusoy (1995): A Survey on the Resource-Constrained Project Scheduling Problem. *IIE Transactions* 27, 574-586.
- Özdamar, L. and G. Ulusoy (1996 a): A Framework for an Interactive Project Scheduling System under Limited Resources. *European Journal of Operational Research* 90, 362-375.
- Özdamar, L. and G. Ulusoy (1996 b): An Iterative Local Constraint Based Analysis for Solving the Resource-Constrained Project Scheduling Problem. *Journal of Operations Management* 14, 193-208.
- Özdamar, L. and G. Ulusoy (1996 c): A Note on an Iterative Forward/Backward Scheduling Technique with a Reference to a Procedure of Li and Willis. *European Journal of Operational Research* 89, 400-407.
- Padman, R. and D.E. Smith-Daniels (1993): Early-Tardy Cost Trade-Offs in Resource Constrained Projects with Cash Flows: An Optimization-Guided Heuristic Approach. *European Journal of Operational Research* 49, 295-311.
- Padman, R.; D.E. Smith-Daniels, and V.L. Smith-Daniels (1997): Heuristic Scheduling of Resource-Constrained Projects with Cash Flows. *Naval Research Logistics* 44, 364-381.
- Papadimitriou, C.H. (1994): Computational Complexity. Addison Wesley, Reading.
- Papadimitriou, C.H. and K. Steiglitz (1982): Combinatorial Optimization: Algorithms and Complexity. Prentice Hall, Englewood Cliffs.
- Partovi, F.Y. and J. Burton (1993): Timing of Monitoring and Control of CPM Projects. *IEEE Transactions on Engineering Management* 40, 68-75.

- Pascoe, T.L. (1966): Allocation of Resources C.P.M. *Revue Francaise de Recherche Opérationnelle* 38, 31-38.
- Patterson, J.H. (1973): Alternate Methods of Project Scheduling with Limited Resources. *Naval Research Logistics Quarterly* 20, 767-785.
- Patterson, J.H. (1976): Project Scheduling: The Effects of Problem Structure in Heuristic Performance. *Naval Research Logistics Quarterly* 23, 95-123.
- Patterson, J.H. (1984): A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem. *Management Science* 30, 854-867.
- Patterson, J.H. and W.D. Huber (1974): A Horizon-Varying, Zero-One Approach to Project Scheduling. *Management Science* 20, 990-998.
- Patterson, J.H. and G.W. Roth (1976): Scheduling a Project under Multiple Resource Constraints: A Zero-One Programming Approach. *AIIE Transactions* 8, 449-455.
- Patterson, J.H.; R. Slowinski, F.B. Talbot, and J. Weglarz (1989): An Algorithm for a General Class of Precedence and Resource Constrained Project Scheduling Problems. In: Slowinski, R. and J. Weglarz (Eds.): *Advances in Project Scheduling*. Elsevier, Amsterdam, 3-29.
- Patterson, J.H.; F.B. Talbot, R. Slowinski, and J. Weglarz (1990): Computational Experience with a Backtracking Alogrithm for Solving a General Class of Precedence and Resource-Constrained Project Scheduling Problems. *European Journal of Operational Research* 49, 68-79.
- Pesch, E. (1999): Lower Bounds in Different Problem Classes of Project Schedules with Resource Constraints. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 53-75.
- Pesch, E. and U. Tetzlaff (1996): Constraint Propagation Based Scheduling of Job Shops. *INFORMS Journal on Computing* 8, 144-157.
- Phillips, S. and M.I. Dessouky (1977): Solving the Project Time/Cost Tradeoff Problem Using the Minimal Cut Concept. *Management Science* 24, 393-400.
- Pirlot, M. (1996): General Local Search Methods. *European Journal of Operational Research* 92, 493-511.
- Pollack-Johnson, B. (1995): Hybrid Structures and Improving Forecasting and Scheduling in Project Management. *Journal of Operations Management* 12, 101-117.
- Popescu, C.M. and S. Pasiphil (1995): Total Float Management in CPM Project Scheduling. *AACE Transactions, C&SM/C.5.1-C&SM/C.5.5*.
- Pritsker, A.A.B.; L.J. Watters, and Ph.M. Wolfe (1969): Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science* 16, 93-108.
- Radermacher (1985/86): Scheduling of Project Networks. *Annals of Operations Research* 4, 227-252.
- Ragsdale, C. (1989): The Current State of Network Simulation in Project Management Theory and Practice. *Omega* 17, 21-25.

- Raz, T. and B. Marshall (1996): Effect of Resource Constraints on Float Calculations in Project Networks. *International Journal of Project Management* 14, 241-248.
- Realff, M.J. and G. Stephanopoulos (1998): On the Application of Explanation-Based Learning to Acquire Control Knowledge for Branch and Bound Algorithms. *INFORMS Journal on Computing* 10, 56-71.
- Reeves, C.R. (1997): Genetic Algorithms for the Operations Researcher. *INFORMS Journal on Computing* 9, 231-250.
- Remer, D.S. and A.P. Nieto (1995): A Compendium and Comparison of 25 Project Evaluation Techniques. Part 1: Net Present Value and Rate of Return Methods. *International Journal of Production Economics* 42, 79-96.
- Ritchie, E. (1985): Network Based Planning Techniques: A Critical Review of Published Developments. In: Rand, G. and R. Eglese (Eds.): *Further Developments in Operations Research*. Pergamon Press, Oxford, 34-56.
- Roy, B. (1964): *Les Problèmes d'Ordonnancement*. Dunod, Paris.
- Roucairol, C. (1996): Parallel Processing for Difficult Combinatorial Optimization Problems. *European Journal of Operational Research* 92, 573-590.
- Russell, A.H. (1970): Cash Flows in Networks. *Management Science* 16, 357-373.
- Russell, R.A. (1986): A Comparison of Heuristics for Scheduling Projects with Cash Flows and Resource Restrictions. *Management Science* 32, 1291-1300.
- Rys, T.; R. Stanek, and W. Ziembla (1994): MIPS: A DSS for Multiobjective Interactive Project Scheduling. *European Journal of Operational Research* 79, 196-207.
- Salewski, F.; A. Schirmer, and A. Drexl (1995): Auftragsorientierte Bildung von Prüferteams für die taktische Personaleinsatzplanung in Wirtschaftsprüfungsgesellschaften. *Zeitschrift für betriebswirtschaftliche Forschung* 47, 211-236.
- Salewski, F.; A. Schirmer, and A. Drexl (1997): Project Scheduling under Resource and Mode Identity Constraints: Model, Complexity, Methods, and Application. *European Journal of Operational Research* 102, 88-110.
- Salkin, H.M. (1975): *Integer Programming*. Addison-Wesley, Reading.
- Sampson, S.E. and E.N. Weiss (1993): Local Search Techniques for the Generalized Resource Constrained Project Scheduling Problem. *Naval Research Logistics* 40, 665-675.
- Savin, D.; S. Alkass, and P. Fazio (1996): Construction Resource Leveling Using Neural Networks. *Canadian Journal of Civil Engineering* 23, 917-925.
- Schirmer, A. (1996 a): New Insights on the Complexity of Resource-Constrained Project Scheduling – A Case of Single-Mode Scheduling. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 390, University of Kiel.
- Schirmer, A. (1996 b): New Insights on the Complexity of Resource-Constrained Project Scheduling – Two Cases of Multi-Mode Scheduling. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 391, University of Kiel.

- Schirmer, A. (1998): Case-Based Reasoning and Improved Adaptive Search for Project Scheduling. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 472, University of Kiel.
- Schirmer, A. and A. Drexel (1997): Allocation of Partially Renewable Resources – Concept, Models, and Application. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 435, University of Kiel.
- Schirmer, A. and S. Riesenbergs (1997 a): Class-Based Control Schemes for Parameterized Project Scheduling Heuristics. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 471, University of Kiel.
- Schirmer, A. and S. Riesenbergs (1997 b): Parameterized Heuristics for Project Scheduling – Biased Random Sampling Methods. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 456, University of Kiel.
- Schmidt, M.J. (1988): Schedule Monitoring of Engineering Projects. *IEEE Transactions on Engineering Management* 35, 108-115.
- Schmidt, R.L. (1993): A Model for R&D Selection with Combined Benefit, Outcome and Resource Interactions. *IEEE Transactions on Engineering Management* 40, 403-410.
- Schmidt, R.L. and J.R. Freeland (1992): Recent Progress in Modeling R&D Project Selection. *IEEE Transactions on Engineering Management* 39, 189-201.
- Schneider, W.G. and D. Hieber (1997): Software zur ressourcenbeschränkten Projektplanung. Technical Report WIOR 494, University of Karlsruhe.
- Schniederjans, M.J. and R. Santhanam (1993): A Multi-Objective Constrained Resource Information System Project Selection Method. *European Journal of Operational Research* 70, 244-253.
- Scholl, A. (1999): Balancing and Sequencing of Assembly Lines. 2nd Ed., Physica, Heidelberg.
- Scholl, A. and R. Klein (1997): SALOME: A Bidirectional Approach for Assembly Line Balancing. *INFORMS Journal on Computing* 9, 319-335.
- Scholl, A. and R. Klein (1999 a): Balancing Assembly Lines Effectively – A Computational Comparison. *European Journal of Operational Research* 114, 51-60.
- Scholl, A. and R. Klein (1999 b): ULINO: Optimally Balancing U-Shaped JIT Assembly Lines. *International Journal of Production Research* 37, 721-736.
- Scholl, A.; R. Klein, and W. Domschke (1998): Pattern Based Vocabulary Building for Effectively Sequencing Mixed-Model Assembly-Lines. *Journal of Heuristics* 4, 359-381.
- Scholl, A.; R. Klein, and Ch. Jürgens (1997 a): BISON: A Fast Hybrid Procedure for Solving the One-Dimensional Bin Packing Problem. *Computers & Operations Research* 24, 627-645.
- Scholl, A.; G. Krispin, R. Klein, and W. Domschke (1997 b): Branch and Bound – Optmieren auf Bäumen: je beschränkter, desto besser. *c't – Magazin für Computer Technik*, Issue 10, 336-345.
- Scholl, A. and S. Voß (1996): Simple Assembly Line Balancing - Heuristic Approaches. *Journal of Heuristics* 2, 217-244.

- Schönert, S. (1998): Kommunikation über alle Kanäle. *Office Management*, Issue 5, 16-17.
- Schott, E.; Ch. Campana, and M. Ilgner (1998): Mehr Tempo mit dem Internet. *Office Management*, Issue 2, 44-47.
- Schrage, L. (1970): Solving Resource-Constrained Network Problems by Implicit Enumeration – Nonpreemptive Case. *Operations Research* 18, 263-278.
- Schrage, L. and K.R. Baker (1978): Dynamic Programming Solution of Sequencing Problems with Precedence Constraints. *Operations Research* 26, 444-449.
- Schultz, V. (1995): Projektkostenschätzung – Kostenermittlung in frühen Phasen von technischen Auftragsprojekten. Gabler, Wiesbaden.
- Schwarze, J. (1994): Netzplantechnik. 7th Ed., Verlag Neue Wirtschaftsbriefe, Herne.
- Schwindt, Ch. (1995): ProGen/max: A New Problem Generator for Different Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Technical Report WIOR 449, University of Karlsruhe.
- Schwindt, Ch. (1996): Generation of Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Technical Report WIOR 516, University of Karlsruhe.
- Schwindt, Ch. (1998 a): A Branch-and-Bound Algorithm for the Resource-Constrained Project Scheduling Problem Subject to Temporal Constraints. Technical Report WIOR 544, University of Karlsruhe.
- Schwindt, Ch. (1998 b): Generation of Resource-Constrained Project Scheduling Problems Subject to Temporal Constraints. Technical Report WIOR 543, University of Karlsruhe.
- Schwindt, Ch. (1998 c): Verfahren zur Lösung des ressourcenbeschränkten Projektdauerminimierungsproblems mit planungsabhängigen Zeitfenstern. Shaker, Aachen.
- Seibert, J.E. and G.W. Evans (1991): Time-Constrained Resource-Leveling. *Journal of Construction Engineering* 117, 503-520.
- Selcuk Erenguc, S. and O. Icmeli Tukel (1999): Integrating Quality as a Measure of Performance in Resource-Constrained Project Scheduling Problems. In: Weglarz, J. (Ed.): *Handbook on Recent Advances in Project Scheduling*. Kluwer, Dordrecht, 433-450.
- Sepil, C. (1994): Comment on Elmaghraby and Herroelen's "The Scheduling of Activities to Maximize the Net Present Value of Projects". *European Journal of Operational Research* 73, 185-187.
- Sepil, C. and N. Ortac (1997): Performance of the Heuristic Procedures for Constrained Projects with Progress Payments. *Journal of the Operational Research Society* 48, 1123-1130.
- Serafini, P. and M.G. Speranza (1994): A Decomposition Approach in a DSS for a Resource Constrained Scheduling Problem. *European Journal of Operational Research* 79, 208-219.

- Shaffer, L.R.; J.B. Ritter, and W.L. Meyer (1965): The Critical Path Method. McGraw-Hill, New York.
- Shipley, M.F.; A. De Korvin, and K. Omer (1996): A Fuzzy Logic Approach for Determining Expected Values: A Project Management Application. *Journal of the Operational Research Society* 47, 562-569.
- Shewchuck, J. and T.C. Chang (1995): Resource-Constrained Job Scheduling with Recyclable Resources. *European Journal of Operational Research* 81, 364-375.
- Shtub, A.; J.F. Bard, and S. Globerson (1994): Project Management – Engineering, Technology, and Implementation. Prentice Hall, Englewood Cliffs.
- Simpson, W.P. and J.H. Patterson (1996): A Multiple-Tree Search Procedure for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research* 89, 525-542.
- Skutella, M. (1998): Approximation and Randomization in Scheduling. Unpublished Ph.D. Thesis, Department of Mathematics, Technological University of Berlin.
- Slowinski, R. (1977): Optimal and Heuristic Procedures for Project Scheduling with Multiple Constrained Resources – A Survey. *Foundations of Control Engineering* 2, 33-49.
- Slowinski, R. (1978): A Node Ordering Heuristic for Network Scheduling under Multiple Resource Constraints. *Foundations of Control Engineering* 3, 19-27.
- Slowinski, R. (1981): Multiobjective Network Scheduling with Efficient Use of Renewable and Nonrenewable Resources. *European Journal of Operational Research* 7, 265-273.
- Slowinski, R. (1988): Multiobjective Project Scheduling under Multiple-Category Resource Constraints. In: Slowinski, R. and J. Weglarz (Eds.): *Advances in Project Scheduling*, Elsevier, Amsterdam, 151-167.
- Slowinski, R.; B. Soniewicki, and J. Weglarz (1994): DSS for Multiobjective Project Scheduling. *European Journal of Operational Research* 79, 220-229.
- Smith-Daniels, D.E. and N.J. Aquilano (1987): Using a Late-Start Resource-Constrained Project Schedule to Improve Project Net Present Value. *Decision Sciences* 18, 369-387.
- Smith-Daniels, D.E.; R. Padman, and V.L. Smith-Daniels (1996): Heuristic Scheduling of Capital Constrained Projects. *Journal of Operations Management* 14, 241-254.
- Smith-Daniels, D.E. and V.L. Smith-Daniels (1987): Maximizing the Net Present Value of a Project Subject to Materials and Capital Constraints. *Journal of Operations Management* 5, 33-45.
- Sondergeld, L. and S. Voß (1997): Optimierung mit dem Ameisen-System – Eine kooperative heuristische Strategie zur Lösung komplexer Optimierungsprobleme. *WiSt – Wirtschaftswissenschaftliches Studium* 26, 568-573.
- Soroush, H.M. (1994): The Most Critical Path in a PERT Network. *Journal of the Operational Research Society* 45, 287-300.
- Souder, W.E. (1984): Project Selection and Economic Appraisal. Van Nostrand Reinhold, New York.

- Speranza, G.M. and C. Vercellis (1993): Hierarchical Models for Multi-Project Planning and Scheduling. *European Journal of Operational Research* 64, 312-325.
- Spinner, M.P. (1997): *Project Management – Principles and Practices*. Prentice-Hall, Englewood Cliffs.
- Sprecher, A. (1994): *Resource-Constrained Project Scheduling – Exact Methods for the Multi-Mode Case*. Springer, Berlin.
- Sprecher, A. (1997): Solving the RCPSP Efficiently at Modest Memory Requirements. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 425, University of Kiel.
- Sprecher, A. (1998): Non-Equivalent Search Strategies for Resource-Constrained Project Scheduling. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 493, University of Kiel.
- Sprecher, A. and A. Drexl (1996): Minimal Delaying Alternatives and Semi-Active Timetabling in Resource-Constrained Project Scheduling. *Manuskripte aus den Instituten für Betriebswirtschaftslehre* 426, University of Kiel.
- Sprecher, A. and A. Drexl (1998): Multi-Mode Resource-Constrained Project Scheduling by a Simple, General and Powerful Sequencing Algorithm. *European Journal of Operational Research* 107, 431-450.
- Sprecher, A.; S. Hartmann, and A. Drexl (1997): An Exact Algorithm for Project Scheduling with Multiple Modes. *OR Spektrum* 19, 195-203.
- Sprecher, A.; R. Kolisch, and A. Drexl (1995): Semi-Active, Active and Nondelay Schedules for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research* 80, 94-102.
- Stinson, J.P.; E.W. Davis, and B.M. Khumawala (1978): Multiple Resource-Constrained Scheduling Using Branch and Bound. *AIIE Transactions* 10, 252-259.
- Sunde, L. and S. Lichtenberg (1995): Net-Present-Value Cost/Time Tradeoff. *International Journal of Project Management* 13, 45-49.
- Sung, C.S. and S.K. Kim (1994): A Project Activity Scheduling Problem with Net Present Value Measure. *International Journal of Production Economics* 37, 177-187.
- Syslo, M.M. (1984): On the Computational Complexity of the Minimum-Dummy-Activity Problem in a PERT Network. *Networks* 14, 37-45.
- Takamoto, M.; N. Yamada, Y. Kobayashi, and H. Nonaka (1995): Zero-One Programming Algorithm for Resource Leveling of Manufacturing Process Schedules. *Systems and Computers in Japan* 26, 68-76.
- Talbot, F.B. (1982): Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case. *Management Science* 28, 1197-1210.
- Talbot, F.B. and J.H. Patterson (1978): An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems. *Management Science* 24, 1163-1175.
- Tavakoli, A. (1990): Effective Progress Scheduling and Control for Construction Projects. *Journal of Management in Engineering* 6, 87-98.

- Tavakoli, A. and R. Riachi (1990): CPM Use in ENR Top 400 Contractors. *Journal of Management in Engineering* 6, 282-295.
- Tavares, L.V. (1986): Multicriteria Scheduling of a Railway Renewal Program. *European Journal of Operational Research* 25, 395-405.
- Taylor, B.W.; L.J. Moore, and E.R. Clayton (1982): R&D Project Selection and Man-power Allocation with Integer Nonlinear Goal Programming. *Management Science* 28, 1149-1158.
- Thamhain, H.J. (1993): Effective Leadership for Building Project Teams, Motivating People, and Creating Optimal Organizational Structures. In: Dinsmore, P.C. (Ed.): *The AMA Handbook of Project Management*. Amacom, New York, 248-258.
- Thesen, A. (1976): Heuristic Scheduling of Activities under Resource and Precedence Restrictions. *Management Science* 23, 412-422.
- Thesen, A. (1977): Measures of Restrictiveness of Project Networks. *Networks* 7, 193-208.
- Thomas, W. (1969): Four Float Measures for Critical Path Scheduling. *Journal of Industrial Engineering* 1, Issue 10, 19-23.
- Thomas, P.R. and S. Salhi (1997): An Investigation into the Relationship of Heuristic Performance with Network-Resource Characteristics. *Journal of the Operational Research Society* 48, 34-43.
- Thomas, P.R. and S. Salhi (1998): A Tabu Search Approach for the Resource Constrained Project Scheduling Problem. *Journal of Heuristics* 4, 123-139.
- Thomassen, O.B. and L. Butterfield (1993): Combining Risk Management and Resource Optimization in Project Management Software. *Cost Engineering* 35, Issue 8, 19-24.
- Tsai, Y.W. and D.D. Gemmill (1998): Using Tabu Search to Schedule Activities of Stochastic Resource-Constrained Projects. *European Journal of Operational Research* 111, 129-141.
- Tsubakitani, S. and R.F. Deckro (1990): A Heuristic for Multi-Project Scheduling with Limited Resources in the Housing Industry. *European Journal of Operational Research* 49, 80-91.
- Ulusoy, G. and L. Özdamar (1989): Heuristic Performance and Network/Resource Characteristics in Resource-Constrained Project Scheduling. *Journal of the Operational Research Society* 40, 1145-1152.
- Ulusoy, G. and L. Özdamar (1994): A Constraint-Based Perspective in Resource-Constrained Project Scheduling. *International Journal of Production Research* 32, 693-705.
- Ulusoy, G. and L. Özdamar (1995): A Heuristic Scheduling Algorithm for Improving the Duration and Net Present Value of a Project. *International Journal of Operations & Production Management* 15, 89-98.

- Valadares Tavares, L.; J.A. Antunes Ferreira, and J. Silva Coelho (1998): On the Optimal Management of Project Risk. *European Journal of Operational Research* 107, 451-469.
- Van Groenendaal, W.J.H. (1998): Estimating NPV Variability for Deterministic Models. *European Journal of Operational Research* 107, 202-213.
- Vercellis, C. (1994): Constrained Multi-Project Planning Problems: A Lagrangean Decomposition Approach. *European Journal of Operational Research* 78, 267-275.
- Vose, D. (1996): Quantitative Risk Analysis: A Guide to Monte Carlo Simulation Modeling. Wiley, Chichester.
- Voß, S. (1993): Intelligent Search. Manuscript, TH Darmstadt.
- Voß, S. (1995): Solving Quadratic Assignment Problems Using the Reverse Elimination Method. In: Nash, S.G. and S. Sofer (Eds.): *The Impact of Emerging Technologies on Computer Science and Operations Research*. Kluwer, Dordrecht, 281-296.
- Wasil, E.A. and A.A. Assad (1988): Project Management on the PC: Software, Applications and Trends. *Interfaces* 18, 75-84.
- Watts, K.M. and J.C. Higgins (1987): The Use of Advanced Management Techniques in R&D. *Omega* 15, 21-29.
- Weglacz, J. (1981): Project Scheduling with Continuously Divisible, Doubly-Constrained Resources. *Management Science* 27, 1040-1053.
- Weglacz, J. (1989): Project Scheduling under Continuous Processing Speed vs. Resource Amount Activity Models. In: Slowinski, R. and J. Weglarz (Eds.): *Advances in Project Scheduling*, Elsevier, Amsterdam, 273-295.
- Weglacz, J.; J. Blazewicz, W. Cellary, and R. Slowinski (1977): ALGORITHM 520: An Automatic Revised Simplex Method for Constrained Resource Network Scheduling. *ACM Transactions on Mathematical Software* 3, 295-300.
- Weiss, E.N. (1988): An Optimization Based Heuristic for Scheduling Parallel Project Networks with Constrained Renewable Resources. *IIE Transactions* 20, Issue 2, 137-143.
- Wheelwright, S.C. and S. Makridakis (1985): Forecasting Methods for Management. 4th Ed., Wiley, Chichester.
- Whitehouse, G.E. and J.R. Brown (1979): GENRES: An Extension of Brooks Algorithm for Project Scheduling with Resource Constraints. *Computers and Industrial Engineering* 3, 261-268.
- Wiest, J.D. (1964): Some Properties of Schedules for Large Projects with Limited Resources. *Operations Research* 12, 395-418.
- Wiest, J.D. (1967): A Heuristic Model for Scheduling Large Projects with Limited Resources. *Management Science* 13, 359-377.
- Wiest, J.D. (1981): Precedence Diagramming Method: Some Unusual Characteristics and their Implications for Project Managers. *Journal of Operations Management* 1, 121-130.

- Wiest, J.D. and F.K. Levy (1977): A Management Guide to PERT/CPM. 2nd Ed., Prentice Hall, Englewood Cliffs.
- Wildausky, A.B. (1986): Budgeting: A Comparative Theory of Budgetary Processes. Transaction Books, New Brunswick.
- Williams, G.S. (1978): Resource Scheduling for Project Control. Power Engineering, Issue 4, 66-69.
- Williams, T. (1995 a): A Classified Bibliography of Recent Research Relating to Project Risk Management. European Journal of Operational Research 85, 18-38.
- Williams, T. (1995 b): What are PERT Estimates? Journal of the Operational Research Society 46, 1498-1504.
- Willis, R.J. (1981): A Note on the Generation of Project Network Diagrams. Journal of the Operational Research Society 32, 235-238.
- Willis, R.J. (1982): The Definition of Float in Resource-Constrained Project Schedules. Bulletin of the Australian Society of Operational Research 1, Issue 2, 5-7.
- Willis, R.J. (1985): Critical Path Analysis and Resource Constrained Project Scheduling – Theory and Practice. European Journal of Operational Research 23, 149-155.
- Woodruff, D.L. and E. Zemel (1993): Hashing Vectors for Tabu Search. Annals of Operations Research 41, 123-128.
- Woodworth, B.M. (1993): A Statistical Evaluation of the Impact of Limited Resources on Project Scheduling. Cost Engineering 35, Issue 2, 25-32.
- Woodworth, B.M. and S. Shanahan (1988): Identifying the Critical Sequence in a Resource Constrained Project. Project Management 6, 90-96.
- Woodworth, B.M. and Ch.J. Willie (1975): A Heuristic Algorithm for Resource Leveling in Multi-Project, Multi-Resource Scheduling. Decision Sciences 6, 525-540.
- Yang, K.-K. and C.-C. Sum (1993): A Comparison of Resource Allocation and Activity Scheduling Rules in a Dynamic Multi-Project Environment. Journal of Operations Management 11, 207-218.
- Yang, K.-K.; F.B. Talbot, and J.H. Patterson (1993 a): Scheduling a Project to Maximize its Net Present Value: An Integer Programming Approach. European Journal of Operational Research 64, 188-198.
- Yang, K.-K.; L.C. Tay, and C.C. Sum (1995): A Comparison of Stochastic Priority Rules for Maximizing Project Net Present Value. European Journal of Operational Research 85, 327-339.
- Yang, T.; J.P. Ingnizio, and J. Song (1989): An Exchange Heuristic Algorithm for Project Scheduling with Limited Resources. Engineering Optimization 14, 189-205.
- Yang, T.; J.P. Ingnizio, J. Yoo, and D. Santos (1993 b): A Modified Exchange Heuristic for Resource-Constrained Scheduling. Engineering Optimization 20, 303-321.
- Yau, C. and E. Ritchie (1988): A Linear Model Estimating Project Resource Levels and Target Completion Times. Journal of the Operational Research Society 39, 855-862.

- Yau, C. and E. Ritchie (1990): Project Compression: A Method for Speeding Up Resource Constrained Projects Which Preserves the Activity Schedule. European Journal of Operational Research 49, 140-152.
- Yoo, J.; T. Yang, and J.P. Ingnizio (1995): An Exchange Heuristic for Resource Constrained Scheduling with Consideration Given to Opportunities for Parallel Processing. Production Planning and Control 6, 140-150.
- Younis, M.A. and B. Saad (1996): Optimal Resource Leveling of Multi-Resource Projects. Computers and Industrial Engineering 31, 1-4.
- Zamani, R. and L.-Y. Shue (1998): Solving Project Scheduling Problems with a Heuristic Learning Algorithm. Journal of the Operational Research Society 49, 709-716.
- Zhan, J. (1992): Calendarization of Time Planning in MPM Networks. Zeitschrift für Operations Research 36, 423-438.
- Zhan, J. (1994): Heuristics for Scheduling Resource-Constrained Projects in MPM Networks. European Journal of Operational Research 76, 192-205.
- Ziegler, H. (1985): Minimal and Maximal Floats in Project Networks. Engineering Costs and Production Economics 9, 91-97.
- Zimmermann, J. (1997): Heuristics for Resource-Levelling Problems in Project Scheduling with Minimum and Maximum Time Lags. Technical Report WIOR 491, University of Karlsruhe.
- Zimmermann, J. and H. Engelhardt (1998): Lower Bounds and Exact Methods for Resource-Levelling Problems. Technical Report WIOR 517, University of Karlsruhe.

Index

A

additive bounding 137
algorithm 92
 evolutionary 190, 210
 exponential time 94
 genetic 210
 polynomial time 93
ant system 191
ascent method 133
aspiration criterion 197
 global, local 197
aspiration-by-default 197, 207
attribute 194
 complementary 194

B

backward pass 24, 44, 78, 115
 modified 91, 141, 176
bound
 argument (*see* lower bound)
 lower (*see* lower bound)
 upper 78, 218, 219
bounding 218, 226
branch 215
branch and bound 214
 scattered 240
branching 215
branching scheme 215
 delaying alternatives 256
parallel 222, 226, 253
schedule scheme 258
serial 254
break-off rule
 dynamic 246, 250
 static 246
budgeting 19, 68

C

candidate list
 global 217
 local 216
strategy 196, 202

completion phase 65
complexity
 index 265
 measure 263
 theory 92
conceptual phase 3, 5, 64
configuration management 18, 28
control 27, 55, 69
 cost 28, 60
 policy 56
 schedule 28, 56
core time 145, 228
cost estimation 10
CPM (critical path method) 24, 43, 66
crashing 21
critical path analysis (*see* CPM)
cut solution 243
 dynamic generation 244
 static generation 244, 248
cutting planes 130
cycling 193

D

decision point 171, 222
 smallest possible 225
decision problem 94
decision support system 14, 109
definition phase 3, 15, 65
destructive improvement 136, 141, 159
dissolution 31
diversification 198, 207, 246, 251
dominance rule 220, 241
 active schedule 230
 extended schedule storing 235
 semi-active schedule 229
 simple schedule storing 235
 storing 221, 232, 256, 257
 supersession 231
 transformation 220, 229, 231
due date 90, 204

E

early starting time approach 49
 earned value analysis 28, **61**, 70
 economic analysis 8
enumeration (*see branching*)
 evaluation 30
 event 41
 execution phase 4, **26**, 68

F

feasibility study 7
 financial appraisal methods 9
 finishing time
 earliest, latest **43**, 78
 scheduled 163
 float (*see slack time*)
 follower (*see successor*)
 forecasting 7, 30
 forward pass 24, **44**, 78, 114
 modified 91, 141, 164, 222

G

Gantt chart 24, **49**, 66, 69
 capacity-oriented 52
 extensions 49
 progress **56**, 70
 GERT (graphical evaluation and review technique) 11
 GRCPSP (generalized resource-constrained project scheduling problem) 67, **89**

H

hashing function 205
 head **124**, 157
 heuristic 161
 adaptive sampling 189
 biased random sampling 189
 constructive 161
 multi-pass priority-rule based 187
 multi-planning directions 188
 multi-priority rule 188
 priority-rule based 167
 regret based biased random sampling 189

sampling 188
 steepest descent 192
 steepest descent / mildest ascent 193

I

improvement method 161, 190
 incompatible pair 76
 partially 153
 instance 93
 intensification **197**, 251

J

job 23, **34**
 active 164
 available 164
 backward available 176
 backward eligible 177
 critical 46
 dummy 38, **42**
 duration 37, **75**
 eligible 79, **164**, 222
 Hammock 69
 list 200
 state preserving 100

L

Lagrangean
 multiplier 132
 relaxation **132**, 137
 left shift
 global **165**, 230
 local **165**, 230
 linear responsibility chart 23, 66, 68
 local lower bound method (LLBM) **216**, 224
 lower bound
 bin packing (LBB1, LBB2) 118
 capacity (LBC2) **115**, 148, 152, 227
 constructive 114
 critical path (LBC1) **114**, 152, 222
 critical sequence (LBC3) **116**, 152
 extended node packing (LBN2) **122**, 156
 generalized node packing (LBN3) **123**, 156
 global 216
 local **216**, 218, 224

node packing (LBN1) 121, 148, 153
 one-machine (LBM2) 126, 157, 227
 parallel-machine (LBM1) 125, 157
 precedence (LBP1, LBP2) 128, 158
 simple 114, 150
 two-machine (LBM3) 127, 158, 227
 lower bound argument (*see* lower bound)
 lower bound method (*see* lower bound)
 LP-relaxation 130, 134

M

make-to-order production 77
 maximum time lag 41, 99, 136
 memory
 explicit 195, 205
 frequency based 197, 246
 recency based 194, 206
 meta-heuristic 190
 method
 cancellation sequence 195
 reverse elimination 196
 milestone 17
 plan 18, 20
 planning 17, 34

minimum time lag 40, 75, 91
 finish-to-finish 40
 finish-to-start 40, 141
 start-to-finish 40
 start-to-start 40, 149
 monitoring 19, 27, 69
 move 191, 201
 admissible 193
 backward shift 201
 diversifying 207
 forward shift 201
 swap 201
 tabu active 193, 194, 206
 MPM (metra-potential method) 41, 99
 multiple modes 96
 multi-project planning 70

N

neighbor 191
 neighborhood 191, 198

net present value 9, 21, 106
 network
 activity-on-arc 18, 23, 41
 activity-on-node 23, 37, 66, 75
 complexity 265
 finish-to-start 75, 77
 start-to-start 75, 90
 neural network 191
 node
 ancestor 215
 descending 215
 leaf 215
 root 215
 NP-complete problem 94
 NP-hard problem 95
 in the strong sense 95

O

objective function
 regular 166
 optimization problem 94
 order of a function 93
 organizational breakdown structure 23, 28, 66, 68

P

path 75
 critical 45, 114
 length 75
 PDM (precedence diagramming method)
 40, 90
 period 37, 78
 PERT (program evaluation and review technique) 11
 phase model 4, 17
 planning
 backward 175
 bidirectional 178
 planning phase 3, 22, 65
 precedence relationship 23, 37
 direct, immediate 38, 75
 finish-to-finish 40, 91
 finish-to-start 37, 75, 91
 indirect, transitive 76

- redundant 39, 76
- start-to-finish 40, 91
- start-to-start 40, 75, 91
- predecessor 38, 75
- preemption 96
- priority rule 181
 - composite 181, 184
 - critical path based 181, 183
 - dynamic 181
 - network based 181, 183
 - regret based 183, 185
 - resource based 181, 184
 - static 181
- problem 93
 - bin packing 117, 225
 - complexity 92
 - flow shop scheduling 77
 - job shop scheduling 77, 95, 117, 165
 - multi-period knapsack 14
 - net present value 21, 106
 - resource investment 105
 - resource leveling 104
 - restricted 137
 - set covering 134
 - simple assembly line balancing 90, 225
 - time-constrained project scheduling 103
 - time-cost trade-off 21
 - type 93
 - weighted tardiness 108
 - weighted-node packing 120
- process organization 17
- product life cycle 2
- PROGRESS 221
- progress plot 57
- project 1
 - calendar 66
 - control (*see* control)
 - life cycle 2
 - management software 63, 167
 - manager 17
 - network (*see* network)
 - organization 16
 - planning 22, 33
 - selection 12
 - specification 15
 - stakeholder 4, 17
 - team 17
- project scheduling 43, 66
 - resource-constrained 26, 52
 - time-constrained 26, 54, 103
- Q**
- quality management 18, 29
- R**
- RCPSP (resource-constrained project scheduling problem) 53, 77
- reduction
 - by core times 144, 160
 - by forward and backward pass 141, 159
 - by precedence 143, 160
 - by subprojects 141, 160
- reduction rule 219, 228
 - core time 228
- reduction technique 138, 141, 159
- region 244
 - swapping 245, 249
- release date 90, 204
- reporting 19, 27, 68
- resource
 - allocation 24, 50, 67
 - availability profile 150
 - compatible set 87
 - conflict 25, 50, 79
 - constraints 24, 50, 77
 - factor 265
 - incompatible set 76, 86
 - leveling 26
 - loading 50
 - loading profile 25, 51, 67
 - residual availability 165
 - strength 266
 - usage 75
- resource type 67
 - doubly-constrained 25, 97
 - non-renewable 25, 97
 - partially renewable 102
 - renewable 25, 50, 75, 77, 97
- RETAPS 198
- risk analysis 10

S

sample size 188

sampling (*see* heuristic)

SCATTER 247

schedule 49

active 166, 170, 230

baseline 49

complete 163

feasible 163

non-delay 166, 172, 180

partial 164

semi-active 166, 229

schedule scheme 208, 258

scheduling scheme 169

backward parallel 177

backward serial 176

bidirectional parallel 180

bidirectional serial 178

parallel (PSS) 171, 222

serial (SSS) 169, 200

search

depth-first (DFS) 216

depth-first with complete branching

(DFSB) 216

laser (DFSL) 216

minimal-lower-bound (MLB) 217

minimal-lower-bound with immediate
branching (MLBI) 217

search trajectory 192, 205

selection rule

dynamic 246, 249

static 246

simulated annealing 209

simulation 11

sink 75

slack time 46

free 46

independent 47

safety 47

total 46, 49, 249

solution

representation 200

space 191, 243

source 75

starting time

earliest, latest 43, 78

scheduled 163

statement of work 15, 36

structuring 22, 34, 65

subproblem 215

subtree

swapping 245, 249

successor 38, 75

T

tabu

list 194

management 194, 205

tenure 194

tenure variation 196, 206

tabu search 191, 208, 242

reactive 197, 206

tail 124, 157

start 225

target analysis 240

termination phase 4, 30

time complexity function 93

time point 78

time window 44, 78, 90

topological ordering 43, 75

transitive closure 76

tree

enumeration 215

minimal enumeration 240, 242

reference 247

trend analysis 59, 70

W

work breakdown structure 20, 22, 28, 34,
65, 68

work package 23, 34