Springer's
International Series

ADVANCING THE
STATE-OF-THE-ART

Volume Contributors:

Anurag Agarwal
Javier Alcaraz
Christian Artigues
Francisco Ballestín
Zbigniew Banaszak
Philippe Baptiste
Anna Baron
Jacques Carlier
Selcuk Colak
Jean Damay
Sophie Demassey
Erik Demeulemeester
Bajis Dodin
S. Selcuk Erenguc
Dimitri Golenko-Ginzburg
Aharon Gonik
Willy Herroelen
Toshihide Ibaraki
Piotr Jędrzejowicz
Tamás Kis
Rainer Kolisch
Krzysztof Kurowski
Philippe Laborie
Roel Leus
Concepción Maroto
Konrad Meyer
Marek Mika
Jarek Nabrzyski
Emmanuel Néron
Klaus Neumann
Koji Nonobe
Ariel Oleksiak
Sacramento Quintanilla
Ewa Ratajczak
Christoph Schwindt
Vicente Valls
Stijn Van de Vonder
Grzegorz Waligóra
Jan Węglarz
Jürgen Zimmermann

PERSPECTIVES IN MODERN PROJECT SCHEDULING

edited by
Joanna Józefowska
Jan Węglarz

PERSPECTIVES IN MODERN PROJECT SCHEDULING

PERSPECTIVES IN MODERN PROJECT SCHEDULING

Edited by
JOANNA JÓZEFOWSKA
Poznań University of Technology

JAN WĘGLARZ
Poznań University of Technology

Springer

Joanna Józefowska
Poznań University of Technology

Jan Węglarz
Poznań University of Technology

Library of Congress Control Number: 2006924377

ISBN-10: 0-387-33643-5 (HB) ISBN-10: 0-387-33768-7 (e-book)
ISBN-13: 978-0387-33643-5 (HB) ISBN-13: 978-0387-33768-5 (e-book)

Printed on acid-free paper.

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

springer.com

Contents

List of Figures

List of Tables

Contributing Authors

Anurag Agarwal is an Assistant Professor in the Department of Decision and Information Sciences, Warrington College of Business at the University of Florida, USA. He has a Ph.D. in Information Systems from The Ohio State University.

Javier Alcaraz is a Professor and an Assistant Director of the Department of Applied Statistics, Operations Research and Quality of the Universidad Politécnica de Valencia. As a member of the Operations Research Team (www.upv.es/gio) he conducts research in the field of Project Management, Flexible Production Programming Techniques and Modeling Optimization Problems.

Christian Artigues is an Assistant Professor at the University of Avignon, France, holding in 2005/2006 an invited professor position at the Center for Transportation Research of the University of Montréal, Québec, Canada. His researches involve hybrid CP/OR scheduling methods and proactive/reactive methods for scheduling under uncertainty.

Francisco Ballestín received his Ph.D. in Statistics and Operational Research from the University of Valencia. He is currently serving as an Assistant Professor at the Public University of Navarra. His research efforts have been focused on the study and efficient resolution of the classic RCPSP and some of its generalizations.

Zbigniew Banaszak is a Professor (Ph.D. 1977, Dr. Habil. 1988) at the Institute of Electronics and Computing Science, Technical University of Koszalin. He has been visiting Professor at the Tokyo Institute of Technology, Kuwait University and Université du Québec a Hull. Member of the IFAC Technical Committee on Intelligent Manufacturing and the Committee on Discrete Event and Hybrid Systems.

Philippe Baptiste holds a research position at CNRS and is an Associate Professor at the Ecole Polytechnique, France. He is mainly interested in Scheduling Theory, Constraint Programming and Operations Research.

Anna Baron is a Doctoral Candidate in the Department of Industrial Engineering and Management at the Ben-Gurion University of the Negev. She earned her M.Sc and Ph.D in Economics in 1992 and 1999, respectively, from State Technical University of St.-Petersburg, Russia.

Jacques Carlier is a Professor at the University of Technology of Compiegne, France, and presently Director of the Doctoral program in Computer Science. His research interests include Combinatorial Optimization and Scheduling problems.

Selcuk Colak is currently pursuing a Ph.D. in the Department of Decision and Information Sciences at the University of Florida, USA. He received a B.Sc. degree from Cukurova University, Adana, Turkey in Electrical and Electronics Engineering, an MS degree from University of Florida, Gainesville, FL in Electrical and Computer Engineering.

Jean Damay is a Teaching and Research Assistant at the Blaise Pascal University of Clermont-Ferrand, France. He recently has defended his PhD presenting several methods based on Linear Programming to solve the preemptive and non-preemptive versions of the RCPSP.

Sophie Demassey had a postdoctoral position at the Center for Transportation Research of Montréal, Canada, in 2004/2005 and is currently associated assistant professor at Ecole des Mines de Nantes, France. Her research is situated in the domain of the integration of constraint and linear programming techniques for combinatorial optimization.

Erik Demeulemeester is a Professor of Operations Management at the Research Center for Operations Management of the Faculty of Economics and Applied Economics of K.U.Leuven (Belgium). Together with Willy Herroelen he has co-authored the Kluwer book on Project Scheduling - A Research Handbook published as volume 49 of the International Series in Operations Research & Management Science.

Bajis Dodin is a Professor of Management Science and Operations Management (MS/OM) at the Univ. of California (UC). Currently he is on assignment for the UC as Director of its Education Abroad Center at the American University in Cairo where he holds a full professor appointment at the Management Department. He earned his M.S and Ph.D. in MS/OM from North Carolina State University. He researches and teaches in the areas of Project Management and MS/OM.

S. Selcuk Erenguc is the Associate Dean for Graduate Programs and the PricewaterhouseCoopers Professor of Decision and Information Sciences at the Warrington College of Business, University of Florida. Dr. Erenguc has also served as a consultant for several companies and taught in several international MBA programs.

Dimitri Golenko-Ginzburg is Professor Emeritus at the Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, and Professor at the Department of Industrial Engineering and Management, Academic College of Judea and Samaria. He received his M.A. degree in Mathematics from the Moscow State University in 1958 and his Ph.D degree in Applied Mathematics from the Moscow Physics-Technical Institute (Russia) in 1962. His professional experience includes 45 years of scientific research and academic teaching in leading academic institutions in the former USSR and, after repatriating in 1985, in Israel.

Aharon Gonik is an Associate Professor and Head of Logistic Department at Sapir Academic College and lecturer at Ben-Gurion University of the Negev in the Department of Industrial Engineering and Management. He received his B.Sc in 1963 and M.Sc in 1974 from the Technion - Israel Institute of Technology Haifa, and Ph.D. in 1995 from the Ben-Gurion University of the Negev, Israel.

Willy Herroelen is a Professor of Operations Management at the Research Center for Operations Management of the Faculty of Economics and Applied Economics of K.U.Leuven (Belgium). Together with Erik Demeulemeester he has co-authored the Kluwer book on Project Scheduling - A Research Handbook published as volume 49 of the International Series in Operations Research & Management Science.

Toshihide Ibaraki is currently a Professor of Kwansei Gakuin University, and is an emeritus professor of Kyoto University. His research interest includes discrete optimization and its algorithms, such as branch-and-bound, dynamic programming and metaheuristics.

Piotr Jędrzejowicz is currently the head of Information Systems Department and Dean of the Faculty of Business Administration, Gdynia Maritime University. His research interests include artificial intelligence, decision support systems and information systems. He is the author of more than 200 papers published internationally. He is also an elected member of the Committee for Informatics, Polish Academy of Science.

Tamás Kis has received his PhD in operations research at the Department of Mathematics of the Swiss Federal Institute of Technology at Lausanne, Switzerland. He is currently affiliated with the Computer and Automation Research Institute of the Hungarian Academy of Sciences, where he conducts research on machine and project scheduling and works on various research and development projects.

Rainer Kolisch is a Full Professor for service and operations management at the Technical University of Munich, Germany. He holds a Dr. in operations research and a Dr. habil. in operations management, both from the university of Kiel, Germany.

Krzysztof Kurowski joined in research and development activities conducted at Poznan Supercomputing and Networking Centre in 1999. Results of his research efforts have been successfully presented at many international workshops and conferences, for instance PPAM, IFORS, CCGrid, HPDC, Supercomputing and GGF.

Philippe Laborie is Principal Scientist at ILOG. His main scientific interests include planning, scheduling, supervision and diagnosis of complex systems and more generally, all decision problems dealing with time.

Roel Leus is an Assistant Professor at the Research Center for OR and Business Statistics of the Faculty of Economics and applied economics of K.U.Leuven (Belgium). His research is focused on scheduling under uncertainty and he has published widely in the field.

Concepción Maroto is a Professor at the Department of Applied Statistics, Operations Research and Quality in the Universidad Politécnica de Valencia and a leader of the Operations Research Team (www.upv.es/gio). She is also the Director General for Environmental Management in the Autonomous Government of Valencia.

Konrad Meyer is an Associate of A.T. Kearney, Germany. He holds a Dr. in operations research from the technical university of Darmstadt, Germany.

Marek Mika is an Assistant Professor at the Institute of Computing Science, Poznan University of Technology.

Jarek Nabrzyski holds a position of Application Department manager at Poznan Supercomputing and Networking Center. He was a member of the Steering Committee of Global Grid Forum which he co-funded in 2001. He serves in advisory boards of many international projects, such as CoreGrid (EU), UCoMs (USA). He is also a member of the advisory board of the KISTI Supercomputing Center in Korea.

Emmanuel Néron is an Assistant Professor at Polytech'Tours, France. He is interested in methods for solving scheduling problems and more especially project scheduling problems.

Klaus Neumann is a Professor of Operations Research at the University of Karlsruhe. He earned his Ph.D. degree from the Technical University of Munich and has authored several books and a large number of articles in different areas of Operations Research.

Koji Nonobe is currently a Lecturer of Hosei University. His interest includes combinatorial optimization, metaheuristics, scheduling and their applications.

Ariel Oleksiak currently works in the Application Department of PSNC as a Computer Systems Analyst. He has actively contributed to many Grid-related projects, e.g. in GridLab, SGIGrid, Clusterix and HPC-Europa.

Sacramento Quintanilla is a Professor of Economic Mathematics at the University of Valencia. Her research is mainly focused on developing methods for the efficient resolution of both theoretical scheduling models and real scheduling problems.

Ewa Ratajczak is an Associate Professor in the Information Systems Department, Faculty of Business Administration, Gdynia Maritime University. She has obtained her Ph.D in computer science from the Technical University, Poznan. Her research interests include combinatorial optimization and Internet technologies. She has published internationally more than 20 papers.

Christoph Schwindt is a Professor of Operations Management at the Clausthal University of Technology. His research interests are in project scheduling, production planning in the process industries, and quantitative supply chain management.

Vicente Valls is a Professor of Statistics and Operational Research at the University of Valencia. He holds a Ph.D degree in Mathematics from the University of Valencia. In addition to his academic activities, Dr. Valls has served as a consultant to a variety of firms.

Stijn Van de Vonder is a Research Assistant at the Research Center for Operations Management of the Faculty of Economics and Applied Economics of K.U.Leuven (Belgium). He is finalizing his Ph.D. thesis on proactive/reactive project scheduling.

Grzegorz Waligóra is an Assistant Professor at the Institute of Computing Science, Poznan University of Technology.

Jan Węglarz is a Full Professor and Director of Institute of Computing Science, Poznan University of Technology, as well as a Director of the Poznan Supercomputing and Networking Center. He is a member of the Polish Academy of Sciences.

Jürgen Zimmermann is a Full Professor and Chairman of the Department of Operations Research at Clausthal University of Technology. He earned his PhD from the University of Karlsruhe and is author of several books and articles in different areas of Operations Research.

Preface

Project scheduling, generally speaking, concerns problems of allocating scarce resources over time to perform a given set of activities in a way taking into account a given performance measure (or measures). The resources are meant here as arbitrary means which activities compete for. Thus, project scheduling problems appear in a large spectrum of real-life situations, and, consequently, they have been intensively studied for over fourty years.

In 1999, the multi-author monograph: J. Węglarz (ed.) Project Scheduling - Recent Models, Algorithms and Applications, was published by Kluwer. Since that time several valuable books have appeared in the field, being however of a rather methodological than state-of-the-art character. Thus, we decided to accept the proposal to edit a new book, continuing, in a conceptual sense, the previously cited one.

Although already ancient projects like building the pyramids in Egypt or the Maya temples in Central America definitely required solving non-trivial resource allocation problems, the first general methods like PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method) were developed (or documented) in the late fifties. Since that time a lot of research has been done in this area including optimization techniques as well as other decision support tools helping to define goals, calculate costs, manage risk and motivate people involved in the realization of the project. Many software packages have been developed as well.

The aim of this book is to present the perspectives of this dynamically developing research area. The content is divided in three parts. In Part I a survey of new models of the project management process are proposed. They include an alternative to the well known PERT technique, consideration of disturbances during the project realization, introduction of new constraints like due dates or setup times, as well as a general discussion on classification of resources.

In Part II new algorithms developed to solve the strongly NP-hard resouce constrained project scheduling problem efficiently with acceptable accuracy are presented. New lower bounds for the RCPSP are proposed, followed by smart justification technique and a series of metaheuristics. Finally, a neural network approach is introduced.

Part III is devoted to new areas of applications of the project management models and algorithms, like pharmaceutical research, grid computing, factory pick-up of new cars, batch scheduling in process industries and make-to order (project driven) manufacturing.

The monograph is addressed primarily to researchers (including PhD. and graduate students), educators, and professionals in the field of: operations management, business administration, system analysis, and applied mathematics. However, specialists in other disciplines like civil, computer and industrial engineering in which resource management problems are of vital importance, can also benefit from it.

We thank all the authors for their valuable contributions and fruitful cooperation.

JOANNA JÓZEFOWSKA AND JAN WĘGLARZ

MODELS

Chapter 1

A PRACTICAL AND ACCURATE ALTERNATIVE TO PERT

Bajis Dodin
A. Gary Anderson Graduate School of Management
University of California, Riverside, CA. 92521
bdodin@aucegypt.edu

Abstract Many real world projects can be represented by stochastic activity network (SAN) models, where the duration of some or all of the project activities are, at best, known in probabilistic sense. These models are known in the literature as PERT networks and they are analyzed using the PERT procedure. It has been known for many years that the Project Evaluation and Review Technique (PERT) provides inaccurate information about the project completion time. Quite often this inaccuracy is large enough to render such estimates as not helpful. As a result of this inaccuracy, many improvements since the introduction of PERT in 1959 have been developed. However, in spite of this inaccuracy and the many improvements, PERT procedure continues to be taught and presented in most text books on Project Management. This is due, perhaps, to its simplicity, and ease of its application. In this paper a new alternative is developed that addresses the issues of accuracy and practicality simultaneously in analyzing SAN models. The new procedure is based on a more accurate representation of the distribution function of the project completion time. We first show that the project completion time can in certain instances be accurately represented by a normal distribution, but in many other instances it can not. In these other instances we show that the project completion time can be more accurately represented by an extreme value distribution. Hence, SAN is first characterized as to when we can use the normal distribution and when we can use extreme value distribution. In the first case, PERT estimates are accurate and will be used; however, in the second case a new procedure is developed that is easy to use, and results in more accurate estimates of the project completion time and its statistics. In this paper examples are also provided that illustrate the above characterization of SANs and the accuracy and practicality of the new procedure.

Keywords: Stochastic activity networks, extreme value distribution, normal distribution, project completion time, estimation.

1.1 Introduction

In project management many real world projects can be modeled as acyclic activity networks. In routine type projects with mostly known requirements, the duration of each activity and/or its required resources can be deterministic. In these kinds of projects, the completion time of the project and each of its activities can be easily determined through the use of the longest path method, known also in Project Management (PM) literature as Critical Path Method (CPM), that was introduced and used in 1959. The duration of the project is given by the realization time of the last node in the network, which is the time/duration of the longest path. By contrast, in non-routine projects, such as those of high technology projects, new product development, behavioral networks and many service oriented projects such as communication or transportation networks, the duration of most, if not all, of the activities may not be known (uncertain); at best they may be known in probability. Similarly for the amount of resources required. For a discussion of the nature of the uncertainty the reader is referred to Elmaghraby (2005).

Uncertainty in the duration of some of the activities of the project, such as those of developing a new product, is the norm rather than the exception. The first step in managing such projects is to represent the duration of each such activity by a random variable (r.v.). The second step is to characterize the r.v.. The ideal case is to characterize the r.v. by a probability distribution function (PDF); this allows for calculating the statistics of the r.v. such as the mean value and variance, and hence the calculation of the statistics of the project completion time and its PDF. However, due to the uniqueness of some activities, and lack of experience or historical data with such activities, management relies on the judgment of the experts in that kind of activities.

Judgments come in different forms. Sometimes short of specifying a PDF for the r.v., the judgment is given as a single quantity representing the mean value of the r.v.; sometimes a range between a specified minimum and maximum (two point estimate) is given, and in case of PERT it is given as three point estimate. The use of a single estimate ignores the chance element associated with the realizations of the r.v.. Similarly the use of more than one estimate ignores the chances of the realizations in between these estimates. This led to the need of selecting a PDF to characterize the realizations of the r.v.. In case of two point estimate, the Uniform PDF was suggested; and in case of three point estimate, Beta and Triangular PDF's were suggested; see for example Elmaghraby (2005), Kamburowski (1997), Elmaghraby (1977), and Malcolm et al (1959). Lack of precision in such judgments has led some researchers to reject the use of PDF and, as an alternative, model the activity durations using fuzzy numbers that are normally used to model imprecise information;

see Herroelen and Leus (2005) and Dubois and Prade (1987), Dubois and Prade (1989).

In project management where the activity is uncertain, most of the literature model the activity duration as a random variable with a known probability distribution or with known mean value and variance. These networks are known as Stochastic Activity Networks (SAN). Then they proceed to determine or approximate the project completion time represented by the realization time of the last node in the stochastic activity network, the completion time of any of the project's milestones, or their respective probability distributions. Such a determination can be a problem in most non-trivial SAN's. This problem has led to many investigations that resulted in the development of some practical procedures. For a review of this literature see Herroelen and Leus (2005), Chapter 9 of Demeulemeester and Herroelen (2002), Krishnan and Ulrich (2001), Adlakha and Kulkarni (1989), and Chapter 4 of Elmaghraby (1977). One of these procedures was developed in 1959; it is known in the literature as Project Evaluation and Review Technique (PERT); see Malcolm et al (1959).

PERT procedure assumes that the project activities are independent of each other, and all what it requires is to characterize the activity duration by the derivation of three time estimates. These are the most optimistic duration, denoted by a, the most pessimistic duration, denoted by p, and the most likely duration, denoted by m. These estimates are normally provided by experienced professionals in the kind of work the activity represents. Then these three values are used to determine the mean value of the activity and its variance. The mean is given by the quantity

$$\mu = (a + 4m + p)/6,$$

and the variance is given by

$$\sigma^2 = (p - a)^2/36.$$

PERT does not require the user to specify a PDF for the r.v. representing the activity duration. The Beta distribution that was specified in the original work on PERT, is believed to have been added for the convenience of developing formulas to derive the mean value and variance for the activities; see Elmaghraby (2005), Kamburowski (1997), Williams (1995), and Donaldson(1965). PERT procedure then moves on to determine the critical path, known also as the longest path, exactly as it is in CPM, using the mean values derived above as the duration of the activities. Hence the mean of the longest path is the sum of the mean values of the activities on the critical path, known also as the critical activities. It is clear that the variances of the activities do not play any role in determining the critical/longest path or it's mean. PERT procedure returns to probability and assumes that the completion time of the project is normally distributed. It makes use of the Central Limit Theorem as the completion time of the project is reduced to the sum of the independent random variables on the longest path. The approximating normal distribution has a mean value equal to that of the longest path and a variance equal to the sum of the variances of the activities on that path. Consequently, standard normal distribution table is then easily used to assess the risk of completing the project at any given time.

Since its inception PERT procedure has come under heavy scrutiny and criticism. It is easy to prove that PERT procedure approximations are not accurate: It underestimates the average completion time of the project, and overestimates its PDF. This has led to the development of various alternatives that provide more accurate estimates; see the above reviews. In spite of the inaccuracies in many of PERT estimates, and the existence of these alternatives, PERT procedure continues to be used and taught in business and engineering schools and presented in many text books. The reason perhaps is its simplicity and the availability and familiarity of the standard normal tables. These two factors make many of the alternative methods harder to use for some researchers and most practitioners. In fact many of these alternatives can not be applied to large activity networks. However, in PERT accuracy has been compromised for practicality. This paper deals with the issues of accuracy and practicality in SAN's simultaneously. It is assumed that the mean and variance of the duration of each activity are given. It presents a procedure that provides a more accurate approximation to the mean, variance, and PDF of the project completion time and it is relatively easy to use. This is achieved by using the Extreme Value theory that was first introduced to project management literature by Dodin and Sirvanci (1990).

The paper is organized as follows. In the next section theoretical discussion of the longest path in SAN's is presented, as its duration represents the completion time of the project. Then the problems of PERT are highlighted and used to introduce the rational of the alternative procedure. In Section 16.3 the relationship between the PDF of the project completion time and the structure of the project network and also the underlying PDF's of the activities are discussed. We show that in many instances, this PDF is closer to an extreme value (EV) distribution than it is to a normal distribution. In Section 16.4 we borrow from Dodin and Sirvanci (1990), David (1981) and Galambos (1978) the properties of the EV distribution and how it can be used to approximate the mean value of the project completion time, its variance and PDF. In Section 16.5 we present the new procedure. Section 15.6 is devoted to some computational experiences and a comparison between PERT estimates, those of the EV, and what is obtained from Monte Carlo sampling. Recommendations are the subject of Section 15.7.

1.2 Theoretical discusion and PERT problems

The following notation will be used throughout the paper:

A: Set of activities in the project network.

M: Number of activities (arcs) in the project stochastic activity network.

N: Number of nodes in the stochastic activity network.

Y_{ij}: A random variable denoting the duration of arc $(i, j) \in A$ which starts in node i and ends in node j.

P: The set of all paths in the network.

$Z(k)$: The duration of path $k \in P$.

T: The duration of the longest path which designates the realization time of the last node in the network and also designates the project completion time.

From the definition of T we notice that

$$T = \max_{k \in P}\{Z(k)\}.$$

Therefore the longest path is not unique. In fact each path $k \in P$ can be the longest path but with certain probability. This probability is known in the literature as the criticality index of the path and it is used to rank the paths; see Dodin and Elmaghraby (1985). The PDF of the r.v. T is

$$F(t) = PR(T \leq t) = PR(Z(k) \leq t \text{ for all k} \in P).$$

Hence,
$F(t) \leq PR(Z(k))$ for any one path $k \in P$ including the PERT critical path.
This shows that, independent of the PERT assumption of normality for the PDF of T, approximating $F(t)$ by the PDF of the duration of only one path is an overestimation, i.e. it forms an upper bound on $F(t)$. Examples can be given to show that such an approximation is grossly optimistic. It can be shown that the joint PDF of any combination of paths continues to form an upper bound on $F(t)$. Therefore, the mean value of the duration of any individual path, or a surrogate stochastic activity network consisting of any combination of the P paths of the original SAN continue to form a lower bound on $E(T)$; see Feller (1968), and Esary et al (1967).

It is clear from the above that the inaccuracies embedded in the PERT procedure emanate from:

- Representing the original project network by a surrogate network consisting of a single path; it is the critical path and the use of its mean as the mean of the longest path. This resulted in shifting the location of $F(t)$ downward. Therefore, involving more than one path in estimating the parameters of T should result in more accurate estimates.

- The variances of the activities on this path or any other path play no role in identifying this path or its mean value; consequently, it is desired to involve the variances in estimating the parameters of T.

- The assumption that the approximate PDF of $F(t)$ is normally distributed with mean value and variance equal to those obtained by PERT procedure. It is shown below that PDF of T is not normally distributed as it is the maximum of many random variables.

Table 1.3 shows an illustration of the underestimation of the PERT mean for $E(T)$. Since calculation of the exact value of $E(T)$ for non-trivial SANs is not possible, we used simulation (Monte Carlo sampling) to generate $E(T)$ given in Table 1.3. The table examined networks generated at random; each is simulated using four different sets of distribution functions for the underlying activities. The activities are not identically distributed as the parameters of each activity are generated at random from a specified range for each of the three distributions: normal, uniform and exponential. In all instances the PERT mean underestimated $E(T)$, and sometimes the downward shift is more than one standard deviation. The downward shift in the location of the PDF of T is the main source of error in PERT procedure estimation of $F(t)$. For instance, in the SAN(40,120) under mixed distributions, and assuming normality for PDF of T, under PERT estimate $PR(T \leq 175) = 0.755$, but according to the simulated result $PR(T \leq 175) = 0.44$. This error in estimating $F(t)$ can also be influenced by the assumption of normality.

In the above example it is remarkable that even though the distribution of T is almost normal, the error caused by underestimating the location of the distribution is even worse. Since inaccurate estimates of the mean and the variance cause large errors, accurate determination of these parameters is of primary importance. These led to many improvements on the original PERT estimates; see for instance Elmaghraby (2000), Soroush (1994), Downey (1990), Devroye (1979), Clingen (1964) and Fulkerson (1962). Estimating the mean and variance of T from many of these procedures is not an easy task where in practice, even before probabilities concerning T are computed, estimates for its mean and variance are needed. In this paper extreme value theory, where EV theory is based on the maximum of independent and identically distributed random variables, is applied to obtain better estimates for the mean and variance of the random variable T. It allows for the use of more than one network path and the variances as well. Improvements in estimating μ_T alone provides more accuracy in estimating $F(t)$.

1.3 Probability distribution function of project duration

The PDF of T may not be normally distributed. It depends on the structure of the network and on the underlying distributions. It was first observed by Van Slyke (1963), as he simulated some SANs, that the PDF of T is right skewed; it has a short left tail and a long right tail. The recognition that PDF of T is not normal has led to many studies to calculate, approximate or bound the exact PDF; see for instance Schmidt and Grossman (2000), Iida (2000), Dodin (1985), Sculli (1983), Sigal et al (1979), Shogan (1977), Kleindorfer (1971), Burt and Garman (1971), Martin (1965) and Clark (1961). Dodin and Sirvanci (1990) showed that such a distribution is between a normal and an extreme value (EV) distribution. They showed that for large networks the distribution of T is affected by two convergences: convergence to a normal distribution as the number of activities on each path increases, and convergence to an EV as the number of paths increases and more of them become more independent and identically distributed. As a result it is desirable to be able to characterize the project network as to when the normal or the EV approximation can be used.

To illustrate the dependency of the PDF of T on the network structure and on the underlying distributions of the activities we investigate two extreme cases. The first shows convergence to normal, while the second shows convergence to an EV. The first network consists of one path with M activities. In this case of completely series network, the longest path duration is equivalent to the length of the only path in the network and the Central Limit theorem, as M increases, is applicable. Therefore,

$$T = \sum_{ij} Y_{ij}, \mu_T = \sum_{ij} \mu_{ij}, \text{ and } \sigma_T^2 = \sum_{ij} \sigma^2(ij).$$

This is an extreme case where the PERT method is clearly valid, and $F(t)$ can easily and accurately be determined from the standard normal table. Table 1.1 shows that PERT estimates for the mean and variance of T are identical to those obtained from simulation. Furthermore, the simulation results show that PDF of T converges to normal as M increases regardless the underlying distribution of the activities.

In the second case, assume that there are M activities, all in parallel, i.e. all activities start at the same node and all end at the same node and $P = M$. Therefore we have the case of completely parallel paths, where the longest path duration is equal to the longest activity.

The Central Limit theorem is not applicable for this case even if each $Z(k)$ is normally distributed. It follows from the definition of $T = \max_{k \in P} Z(k)$ and the independence of the P paths that

Table 1.1. Simulation of SAN's where each consists of one path.

Arcs		Normal $\mu \in [5:15]$ $\sigma = 0.2\mu$		Uniform $a \in [0:10]$ $b = a+3$		Exponencial $\lambda \in [0.1:2]$		Mixed	
		PERT	Sim	PERT	Sim	PERT	Sim	PERT	Sim
10	μ	104.44	104.42	80.97	80.85	7.61	7.45	52.93	53.11
	σ	6.82	6.82	5.48	5.61	2.50	2.46	11.76	11.88
20	μ	208.31	208.17	170.25	170.36	42.45	42.33	124.45	124.34
	σ	9.70	10.15	7.75	7.67	13.86	13.77	8.30	8.44
40	μ	414.46	414.19	281.08	280.74	56.81	56.83	292.85	293.12
	σ	13.49	13.34	10.95	11.00	11.22	11.25	11.92	11.82

$$F(t) = PR(Z(k) \leq t \text{ for all } k \in P) = \Pi_{k \in P} F_k(t)$$

which is not normally distributed. If the $Z(k)$'s are independent and identically distributed (iid), then the PDF of T is equal to $[F_k(t)]^P$. If the P parallel paths are iid, then PDF of T converges to EV distribution.

Table 1.2 shows the simulation results for seven structures where P ranges from 5 to 90 parallel and iid paths. Each is simulated for the three specified distributions: Exponential with $\lambda = 1$, gamma with $k = 5$ and $\lambda = 1$, and normal with $\mu = 4$ and $\sigma = 1$. For each case in Table 1.2, four numerical results are calculated; the simulated mean value of T, the standard deviation of T and, as a measure for goodness-of-fit for the distribution of T, two chi-square test statistic values, χ^2 for normal and for EV, are computed for each case. It is very clear that as P increases, PDF converges to an EV regardless the underlying distribution of the individual path. In case of exponential and gamma distributions the convergence is immediate, where in case of normal it is slow.

Figure 1.1 shows a simulated PDF for an activity network with with 40 nodes and 100 activities where each activity has an exponential distribution with λ generated at random from the interval [0.2, 5.0]. It is also observed from Table 1.2 that $E(T)$ shifts to the right if compared with the mean of each path which stays constant. The amount of shift increases as P increases; for $P = 90$ the amount of shift can exceed four times the standard deviation.

Most real world problems fall between the above two extremes. The adequacy of normal approximation in case of a network with one path depends on the number of activities in series and their corresponding PDF's. This is also true for the adequacy of the EV approximation for the PDF of T in the completely parallel network. For most real world projects, the corresponding network may have several paths close in duration competing for the longest path. As a result, the probabilistic mechanism which gives rise to the extreme value

Table 1.2. Simulation results of completely parallel activity networks.

Number of parallel paths	Normal $\mu = 4, \sigma = 1$				Gamma $(k = 5, \lambda = 1)$, $\mu = 5, \sigma = 2.236$				Exponential $(\lambda = 1), \mu = 1, \sigma = 1$			
	$E(T)$	s_T	χ^2_n	χ^2_{EV}	$E(T)$	s_T	χ^2_n	χ^2_{EV}	$E(T)$	s_T	χ^2_n	χ^2_{EV}
5	5.17	0.68	39.3	108.7	7.82	2.09	127.2	10.9	2.27	1.21	303.8	35.5
10	5.54	0.59	42.4	63.4	9.01	2.06	146.3	17.4	2.96	1.27	258.2	11.8
15	5.76	0.55	56.4	61.4	9.65	1.97	123.9	16.3	3.32	1.21	250.1	24.3
20	5.88	0.53	71.6	53.4	10.04	1.90	134.7	18.9	3.64	1.28	212.4	18.4
25	5.99	0.51	64.0	65.7	10.43	1.87	104.8	29.0	3.82	1.27	250.5	19.3
30	6.05	0.50	52.1	57.7	10.68	1.85	141.8	16.0	4.03	1.30	239.9	13.2
90	6.64	0.43	79.1	39.2	12.26	1.83	181.7	24.52	5.11	1.28	210.9	18.3

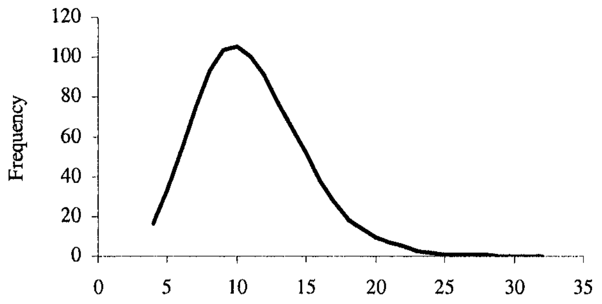


Figure 1.1. Probability density function of T for a SAN with 40 nodes and 100 activities with un-identical exponential distributions.

theory will be in effect for such networks. In the following section properties of the EV distribution are presented.

1.4 Properties of extreme value distribution

The asymptotic distribution of T_m as $m \to \infty$ where the T_m is the maximum of m iid random variables with a PDF $F(t)$ and a density function $f(t)$ has been studied in probability theory; see David (1981) and Galambos (1978). The following theorem is adapted from David (1981).

THEOREM 1.1 *Suppose $F(t)$ is less than 1 for every finite t, is twice differentiable at least for all t greater than some value τ and is such that*

$$\lim_{t \to \infty} \frac{d}{dt}\left(\frac{1 - F(t)}{f(t)}\right) = 0$$

then,

$$\lim_{m \to \infty} PR\{b_m(T_m - a_m) \leq t\} = G(t),$$

holds uniformly for every $t \in (-\infty, \infty)$, where

$$G(t) = exp(-e^{-t}) \quad \text{and}$$

$$F(a_m) = (m-1)/m \quad \text{and} \quad b_m = mf(a_m).$$

It can be shown that a large class of underlying PDF's, including such distributions as the normal and the exponential, satisfies the conditions of the above theorem. The limiting PDF, $G(t)$, given above is known as the standard extreme value distribution. The graph of it's probability density function, $g(t)$, is given in Figure 1.2.

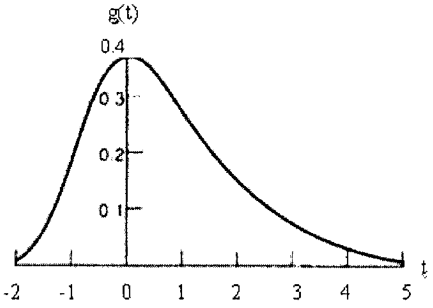


Figure 1.2. Standard extreme value density function where $g(t) = exp[-t - e^{-t}]$.

The general form of the two-parameter EV distribution function is

$$G(t) = exp[-e^{-b(t-a)}]$$

where a is the location parameter (mode) and b is the scale parameter. The mean and the variance of $G(t)$ can be written in terms of a and b as follows:

$$\mu = a + 0.57722/b \quad \text{and} \quad \sigma^2 = \pi^2/(6b^2).$$

The parameters of the limiting EV distribution, location parameter a and scale parameter b, can be estimated from the above theorem where,

$$F(a_m) = (m-1)/m \quad \text{and} \quad b_m = mf(a_m),$$

and $F(t)$ is now assumed to be a normal PDF with mean μ and variance σ^2. The mean value μ and variance σ^2 are those of one of the iid parallel paths in the network that are competing for the longest path. Thus the location parameter, a, can be approximated by a_m, and the scale parameter, b, can be approximated by b_m. Therefore, from the normal probability distribution function we have,

$$\frac{e^{-(a_m-\mu)^2/2\sigma^2}}{\sqrt{2\pi}(a_m-\mu)/\sigma)} = \frac{1}{m}.$$

From Cramer (1964), it follows that

$$a_m = \mu + \sigma[(2\ln m)^{1/2} - (l/2)(\ln\ln m + \ln 4\pi)/(2lnm)^{1/2}],$$

and

$$b_m = \frac{\sqrt{2\ln m}}{\sigma}$$

where m is the number of paths competing for the longest path, i.e. dominating paths, in the network. Therefore, the mean of the limiting EV distribution can be approximated by

$$\mu_{EV} = a_m + 0.57722/b_m$$

and the standard deviation can be approximated by,

$$\sigma_{EV} = \pi/(2.45b_m).$$

It is clear from the expressions for a_m and b_m that the estimations of μ_{EV} and σ_{EV} involve the standard deviation of the longest path, and hence the standard deviations of the activities on that path. The PDF can be approximated by,

$$PR(T \leq t) = G(t) = exp[-e^{-b_m(t-a_m)}].$$

Extreme value theory has been applied successfully to many practical problems; see, for example, Galambos (1978). In its applications, frequently the conditions sufficient for the validity of the limiting extreme value distribution cannot all be verified. Among the conditions which are usually difficult to satisfy are the independence and identically distributed properties of the underlying random variables. Even in these instances, extreme value theory has provided insights and useful results for the problems considered. In many stochastic networks, the distributions of paths are not identical, and since paths may have arcs in common, they are not independent of each other. However, as the size of the stochastic network increases the number of paths increases, dependency between the paths is weakened, and more paths compete for the longest path, i.e become almost iid. As a result, we expect the PDF of T to deviate from normal and move toward the EV distribution.

The theoretical analysis presented above is in agreement with the conclusion derived from the empirical results presented in Dodin and Sirvanci (1990), where it was concluded that when the number of paths competing for the longest path in a network is relatively large and/or the underlying arc distribution is right-skewed, the EV distribution gives a better fit than the normal distribution. Application of the EV approximation may further be justified since, in practice, the distribution of the duration of most activities tends to be right-skewed, i.e. the probability of completing an activity after the most likely time is usually greater than the probability of completing it before that. In the next section the above approximations are used in developing an alternative procedure to PERT.

1.5 Alternative procedure to PERT

It was conclude above that in large stochastic activity networks the PDF of the completion time of the project is influenced by two convergences. The first is convergence to the normal distribution emanating from the fact that each single path consists of many activities; hence it's duration converges to normal distribution. The second is convergence to EV distribution emanating from having many paths in the network competing for the longest path, that are close in duration and close to being independent. If the first influence is stronger, then PERT procedure provides accurate estimates and it will continue to be used; however, if the second influence is stronger, then estimates using extreme value theory are more accurate and they will be used. Consequently, the first step in this procedure is to characterize the stochastic network as to when to use each of the two estimates. The second step in the procedure is to carry out the corresponding estimation for the statistics of T: its mean μ_T, standard deviation σ_T, and PDF, $F(t)$. The following are the steps of the procedure:

1 If the mean and standard deviation of every activity in the network are not given, then calculate them form the given information.

2 Apply CPM method or any of the labeling methods given in Glover et al (1992), using the mean values of the activities, to calculate the means of the two longest paths (in mean) and the corresponding standard deviations. Denote these by μ_1, σ_1, μ_2, and σ_2, respectively.

3 If the first longest path dominates the second longest path, then the number of dominating paths $m = 1$, and the influence of convergence to normal distribution is stronger; go to 4. Otherwise $m = 2$ and go to 5. In theory we say path p dominates path q if

$$PR[Z(p) \geq Z(q)] \geq PR[Z(q) \geq Z(p)] = 1 - PR[Z(p) \geq Z(q)].$$

Calculating the above expression is difficult. Therefore we consider the first longest path as dominating if $\mu_1 - \mu_2 \geq \max\{0.05\mu_1, 0.20\sigma_1\}$, i.e. they are not close in mean.

4 The PDF of T, completion time of the project, is assumed to be normally distributed with mean μ_T approximated by μ_1, and σ_T approximated by σ_1, exactly as it is in PERT, then terminate.

5 Apply Step 2 above, repeatedly if necessary, to determine the value of m, the number of dominating paths (that are close in mean and they compete for the longest path). In this step, first the mean and standard deviation of the third longest path are determined and compared with that of the first longest path for dominance exactly as it is in Step 3. If the new path is dominated, then got to 6; otherwise set $m = m + 1$ and repeat.

6 The average completion time of the project μ_T is approximated by μ_{EV}, and σ_T is approximated by σ_{EV} as they are stated in Section 16.4. To calculate μ_{EV} and σ_{EV}, we use the values of m, μ_1 and σ_1.

In this case the approximation of PDF of T, $F(t)$, as discussed above, depends on the value of m, number of dominating paths, and on the underlying distributions of these paths. For $m \geq 2$, $F(t)$ is expected to be between normal and EV. For small values of m, such as $m \leq 5$, and with each path consisting of many activities we expect $F(t)$ to be closer to normal; hence we use the normal distribution to approximate $F(t)$ but with μ_T and σ_T as calculated in this procedure. Otherwise we use $G(t)$ specified above to approximate $F(t)$.

1.6 Computational experience

As a benchmark, the above procedure was first applied to the stochastic networks presented in Figures 3 and 4 of Dodin and Sirvanci (1990) and for the same distributions. In these SAN's all the activities are identically distributed. The value of m for the network of Figure 3 was 3, and in Figure 4 it was 12. As a result, the procedure derived the same values for μ_T and σ_T. Similarly, the procedure was also applied to the seven networks of Table 1.2 where all paths are iid; EV estimates were identical to those of simulation. In most stochastic project networks, activities are not identically distributed or have the same distribution type, and project networks can have various structures (precedence relations). Therefore the procedure was also tested using activity networks generated at random, and also assigned durations with different probability distribution functions where their parameters are randomly generated from specified ranges for each distribution type.

The procedure was applied to the nine network structures presented in Table 1.3 where the number of nodes range from 10 to 40, and the activities range from 16 to 120. Each network was tested for four different underlying distributions: Normal, uniform, exponential and a mixture of the three. The most realistic (practical) among these is the mixed distribution case. In all 36 combinations the activities are not identically distributed as the corresponding parameters for each activity are generated at random from the ranges specified in Table 1.3. First, in each combination of (network, distribution) each activity is assigned its own parameter/s for that distribution type; for instance in the combination of [network (10, 16) and normal distribution], each of the 16 activities have its

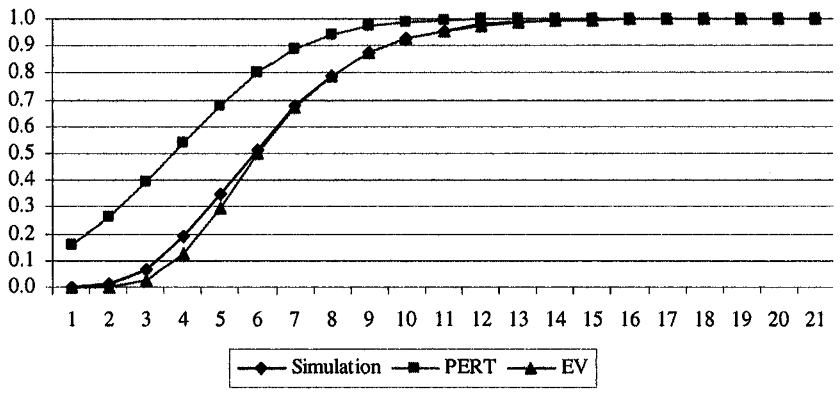


Figure 1.3. Cumulative distributions for the estimates of PERT, simulation and extreme value.

μ generated at random from the interval [5, 15], and $\sigma = 0.2\mu$. Second, the PERT critical path was identified and its estimates of μ_T and σ_T were derived and recorded using CPM. Third, the above procedure was applied and estimates of μ_T and σ_T were recorded. Fourth, to be able to verify the accuracy of the estimated parameters and the resulting $F(t)$ we used Monte Carlo sampling to generate the corresponding parameters and probability distributions for all the combinations. In this case each combination was simulated using a sample of size n, where n is between 1000 and 5000, and estimates of μ_T and σ_T are recorded along with the approximation of $F(t)$. Estimates obtained from simulation are considered the actual (accurate) values of these parameters.

Table 1.3 shows the three estimates for the mean, μ_T and variance, σ_T : PERT, simulation, and new procedure (NP); Table 1.4 shows the number of the m dominating paths as a result of using the above dominance criterion. If in Table 1.4, $m = 1$, then the corresponding values for μ_T and σ_T under the column of NP in Table 1.3 are set equal to those of PERT; otherwise they are calculated using extreme value estimates given in Section 16.4. It is clear from Table 1.3 that PERT estimates for μ_T always bound the actual μ_T from below. It is also clear that whenever $m \geq 2$, extreme value gives more accurate estimates for μ_T than PERT. Consider for instance the cases for 40 nodes and exponential distribution; estimates of EV are superior to those of PERT. Similarly for many others combinations; naturally, some estimates are more accurate than others.

Table 1.3. Three estimates for μ_t and σ_T.: PERT, simulation, and new procedure for various stochastic activity networks (different network structures and different underlying PDF's).

Nodes	Arcs		Normal $\mu \in [5:15]$ $\sigma = 0.2\mu$			Uniform $a \in [0:10]$ $b = a+3$			Exponential $\lambda \in [0.1:2]$			Mixed		
			Simul.	PERT	*New proc.*	Simul.	PERT	*New proc.*	Simul.	PERT	*New proc.*	Simul.	PERT	*New proc.*
	16	μ	47.58	44.68	49.20	34.95	33.05	35.64	13.44	11.00	16.86	23.32	21.71	24.24
		σ	3.33	4.58	3.96	2.65	3.46	3.77	6.87	7.84	8.53	2.80	3.38	3.68
10	23	μ	50.72	48.62	52.29	45.34	43.15	45.74	16.84	15.78	$n=1$	42.03	40.54	43.18
		σ	3.81	4.91	5.34	2.84	3.46	3.77	9.41	9.87		2.92	3.51	3.83
	30	μ	67.75	66.67	$n=1$	58.27	56.85	$n=1$	20.26	17.27	$n=1$	45.23	41.24	48.39
		σ	4.57	5.18		4.09	4.58		9.50	9.94		8.34	9.56	10.40
	30	μ	84.42	82.57	$n=1$	75.52	75.06	$n=1$	21.45	18.25	$n=1$	57.84	56.86	$n=1$
		σ	5.71	6.50		4.27	4.58		9.68	9.88		4.85	5.20	
20	40	μ	94.82	93.52	$n=1$	79.49	78.22	$n=1$	27.36	24.19	$n=1$	63.23	61.69	65.53
		σ	6.22	6.78		4.64	4.90		10.47	11.12		4.84	5.14	5.60
	60	μ	125.70	123.42	$n=1$	101.13	100.37	$n=1$	29.82	25.16	33.11	85.12	82.05	86.42
		σ	7.59	8.03		5.19	5.74		10.10	10.62	11.56	7.24	5.84	6.36
	75	μ	158.11	152.27	161.01	122.56	118.89	124.81	39.77	33.65	42.47	142.48	138.22	143.39
		σ	7.67	8.86	7.66	4.91	6.00	5.19	10.85	11.77	12.82	8.01	6.90	7.51
40	100	μ	179.63	176.56	$n=1$	137.81	135.03	140.06	54.55	41.20	55.71	157.07	154.81	163.27
		σ	7.89	9.26		5.76	6.71	7.30	13.52	14.71	12.72	10.17	11.30	12.30
	120	μ	195.33	187.89	194.92	164.94	160.74	166.09	59.11	41.54	59.09	176.33	169.15	177.54
		σ	7.80	9.39	10.22	6.28	7.14	7.78	13.73	13.74	9.82	8.54	8.51	7.36

Table 1.4. Number of dominating paths determined by the new procedure.

Network	Normal	Uniform	Exponential	Mixed
(10, 16)	3	2	2	2
(10, 20)	2	2	1	2
(10, 30)	1	1	1	2
(20, 30)	1	1	1	1
(20, 40)	1	1	1	2
(20, 60)	1	1	2	2
(40, 75)	3	3	2	2
(40, 100)	1	2	3	2
(40, 120)	2	2	5	3

Examination of Tables 1.3 and 1.4 shows that more than 60% of the cases have more than one dominating path; hence EV estimation for μ_T and σ_T was used. Most of the cases where EV estimation was used are in the large activity networks, such as those of networks with 40 nodes; and also in the cases with mixed distributions. Both cases, very large networks and networks with mixed distributions, tend to produce more paths with close durations that also are close to being independent. Smaller size networks tend to produce one dominating path. This path may share with other paths the same set of activities, hence they are very dependent, and differs from them by only one or few activities with larger duration; hence it remains always dominating. The structure of the networks with 20 nodes produced one strong dominating path; hence PERT estimates were mostly accurate regardless the underlying distributions.

As for the estimate of $F(t)$, this is more sensitive to the underling distributions than the estimates of μ_T or σ_T. If the activity network possesses many probability distributions with large right hand tails, then extreme value estimates are more accurate than normal. Similarly, if the network has large m dominating paths that are very much independent of each other, i.e. do not have many activities in common. Table 1.5 and Figure 1.3 have an example of the three estimates for $F(t)$: Normal based on μ_{PERT}, and σ_{PERT}, extreme value with μ_{EV}, and σ_{EV}, and simulation for the activity network with 40 nodes and 100 activities and different exponential distributions for the activities. It is clear from Figure 1.3 that EV estimates are very close to the simulated $F(t)$, and PERT estimates of $F(t)$ forms a very loose upper bound.

The above estimates are very sensitive to the assumptions of the extreme value theory: existence of many paths that are iid. Therefore, the estimates are sensitive to the number of dominating paths m, their independence and durations. The dependence on m, is clear from the expressions for a_m and b_m. However, the value of m depends on the dominance rule stated in Step 3 of

Table 1.5. Three estimates of F(t): PERT, simulation and extreme value.

t	Simulation	PERT	EV
26.35	0.00	0.16	0.00
31.79	0.02	0.26	0.00
37.23	0.07	0.39	0.03
42.67	0.19	0.54	0.12
48.11	0.35	0.68	0.30
53.55	0.52	0.80	0.50
58.99	0.68	0.89	0.67
64.43	0.79	0.94	0.79
69.87	0.88	0.97	0.87
75.31	0.93	0.99	0.93
80.75	0.96	1.00	0.96
86.19	0.98	1.00	0.97
91.63	0.99	1.00	0.99
97.07	0.99	1.00	0.99
102.51	1.00	1.00	0.99
107.95	1.00	1.00	1.00
113.39	1.00	1.00	1.00
118.83	1.00	1.00	1.00
124.27	1.00	1.00	1.00
129.71	1.00	1.00	1.00
135.15	1.00	1.00	1.00

the procedure. If the difference $\mu_1 - \mu_2$ is enlarged, m may increase; however the m paths will deviate from being identically distributed, and extreme value theory will not apply. Similarly, if the paths are very dependent such sharing many activities, extreme value assumptions will not be satisfied. Therefore, in the above procedure extreme value estimates work well for large networks with many parallel paths that are close in duration.

1.7 Conclusion

This paper deals with the problem of providing an accurate and easy to calculate estimates to the statistics of the project completion time in stochastic activity networks. It demonstrates that the probability distribution function of the project completion time may under certain conditions be normally distributed; but in general it is not, and the estimates for its mean and standard deviation obtained by the PERT procedure are in general not accurate. It also shows that such a probability distribution function may under certain conditions converge to an extreme value distribution. The paper then characterizes the project network as to when the normal or extreme value distributions can be used to estimate the probability distribution of the project completion time and its parameters.

A certain form of stochastic dominance is introduced and used in the above characterization. It uses the mean and standard deviation of the longest path, in mean, in this characterization. If the network has only one dominating path, then PERT procedure continues to provide accurate estimates and it is easy to use. If, however, the network has more than one dominating path, then extreme value theory is used to estimate the mean and standard deviation of the project completion time. Extreme value estimates involve the number of the longest (most dominating paths), mean of the longest path, and its standard deviation. If the number of the dominating paths is large, or the underlying distributions are very rightly skewed, then extreme value distribution is also used to approximate the probability distribution function of the project completion time. Calculation of these statistics using extreme value distribution is relatively easy.

The above procedure was applied to several stochastic networks and was found to be of merit. It provides accurate estimates to μ_T, σ_T, and $F(t)$, and it is relatively easy to apply. The issue that may require further investigation is the development of a more precise and easy to apply method to determine when a certain path in the network dominates another.

References

Adlakha, V. G., and Kulkarni, V. G. (1989). A Classified Bibliography of Research on Stochastic PERT Networks: 1966-1987, *INFOR*, 27(3):272–296.

Burt Jr., J. M. and Garman, M. B. (1971). Conditional Monte Carlo: a Simulation Technique for stochastic network analysis, *Management Science*, 18:207–217.

Clark, C. E. (1961). The Greatest of a Finite Set of Random Variables, *Operations Research*, 9:146–162.

Clingen, C. T. (1964). A Modification M Fulkerson's PERT Algorithm. Letter to the Editor, *Operations Research*, 12:629–632.

Cramer, H. (1946). *Mathematical Models of Statistics*, Princeton University Press, Princeton, NJ.

David, H. A. (1981). *Order Statistics*, 2nd ed. Wiley, New York.

Demeulemeester, E., Herroelen, W. (2002). *Project Scheduling: A Research Handbook*, Kluwer Academic Publishing.

Devroye, L. (1979). Inequalities for the Completion Times of Stochastic PERT Networks, *Mathematics of Operations Research*, 4:441–447.

Dodin, B. M. and Elmaghraby, S. E. (1985). Approximating the Criticality Indices of the Activities in PERT Networks, *Management Science*, 31:207–223.

Dodin, B. M. (1985). Bounding the Project Completion Time Distribution in PERT Networks, *Operations Research*, 24:862–882.

Dodin, B., and Sirvanci, M. (1990). Stochastic Networks and the Extreme Value Distribution, *Computers and Operations Research*, 17(4):397–409.

Donaldson, W. A. (1965). Estimation of the Mean and Variance of a PERT Activity Time, *Operations Research*, 13:382–385.

Downey, P. J. (1990). Distribution-Free Bounds on the Expectation of the Maximum with Scheduling Applications, *Operations Research Letters*, 9:189–201.

Dubois, D. and Prade, H. (1987). Fuzzy Numbers: An Overview, in: *Analysis of Fuzzy Information*, Bezdek, J.C., ed, CRC Press, Boca Raton, pp. 3–39.

Dubois, D. and Prade, H. (1989). Processing Fuzzy Temporal Knowledge, *IEE Transactions on Systems, Man and Cybernetics*, 19(4):729–744.

Elmaghraby, S. E. (1977). *Activity Networks: Project Planning and Control by Network Models*, Wiley and Sons, New York.

Elmaghraby, S. E. (2000). On Criticality and Sesitivity in Activity Networks, *European Journal of Operational Research*, 127:220–238.

Elmaghraby, S. E. (2005). On the fallacy of averages in project risk management, *European Journal of Operational Research*, 165(2):307–313.

Esary, J. D., Proschan, F. and Walkup, D. W. (1967). Association of Random Variables with Applications, *Annals of Mathematical Statistics*, 38:1466–1474.

Feller, W. (1968). *An Introduction to Probability Theory and its Applications*, Vol. I, 3rd ed., John Wiley & Sons, New York.

Fulkerson, D.R. (1962). Expected critical path lengths in PERT networks, *Operations Research*, 10:808–817.

Galambos, J. (1978). *The Asymptotic Theory of Extreme Order Statistics*, John Wiley & Sons, New York.

Glover, F., Klingman, D., and Phillips, N. V. (1992). *Network Models in Optimization and their Applications in Practice*, John Wiley & Sons, New York.

Herroelen, W., and Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials, *European Journal of Operational Research*, 165(2):289–306.

Iida, T. (2000). Computing Bounds on Project Duration Distributions for Stochastic PERT Networks, *Naval Research Logistics*, 47:559–580.

Kamburowski, J. (1997). New Validations of PERT Times, *OMEGA*, 25(3):323–328.

Kleindorfer, G. B. (1971). Bounding Distributions for a Stochastic Acyclic Network, *Operations Research*, 19:1586–1601.

Krishnan, V. and Ulrich, K. T. (2001). Product development decisions: A review of the literature, *Management Science*, 47:1–21.

Malcolm, D.J., Roseboom, J.H., Clark, C.E. and Fazar, W. (1959). Application of a technique for research and development program evaluation, *Operations Research*, 7:646–669.

Martin, J. J. (1965). Distribution of the Time through a Directed Acyclic Network, *Operations Research* 13:46–66.

Schmidt, C. W. and Grossmann, I. E. (2000). The Exact Overall Time Distribution of a Project with Uncertain Task Durations, *European Journal of Operational Research*, 126:614–636.

Sculli, D. (1983). The Completion Time of PERT Networks, *Journal of the Operational Research Society*, 34:155–158.

Shogan, A. W. (1977). Bounding Distributions for a Stochastic PERT Network, *Networks*, 7:359–381.

Sigal, C. E., Pritsker, A. A. B., and Solberg, J. J. (1979). The use of Cutset in Monte Carlo Analysis of Stochastic Networks, *Mathematics and Computers in Simulation*, 21:379–384.

Soroush, H. (1994). Risk Taking in Stochastic PERT Networks, *European Journal of Operational Research*, 67:221–241.

Van Slyke, R M. (1963). Monte Carlo methods and the PERT problem, *Operations Research*, 11:839–860.

Williams, T. M. (1995). What are PERT Estimates?, *Journal of the Operational Research Society*, 46:1498–1504.

Chapter 2

PROACTIVE-REACTIVE PROJECT SCHEDULING TRADE-OFFS AND PROCEDURES

Stijn Van de Vonder, Erik Demeulemeester, Roel Leus, Willy Herroelen
Research Center for Operations Management, K.U.Leuven
Naamsestraat 69, B-3000 Leuven (Belgium)
Email: <first name>.<last name>@econ.kuleuven.be

Abstract The vast majority of the research efforts in project scheduling over the past several years have concentrated on the development of exact and heuristic procedures for the generation of a workable baseline schedule assuming complete information and a static and deterministic environment. During project execution, however, a project may be subject to considerable uncertainty. Proactive-reactive project scheduling deals with uncertainty by creating a baseline schedule that is as much as possible protected against disruptions and by deploying reactive scheduling procedures to revise or reoptimize this schedule when necessary. This chapter focuses on the main principles of proactive-reactive scheduling and dwells on schedule robustness and its measurements. A number of recently developed proactive and reactive scheduling heuristics are described and their working principles are illustrated on a problem example

Keywords: Project scheduling, uncertainty, robustness

2.1 Introduction

The vast majority of the research efforts in project scheduling over the past several years have concentrated on the development of exact and heuristic procedures for the generation of a workable *baseline schedule* (*pre-schedule* or *predictive schedule*) assuming complete information and a static and deterministic environment. Most often the baseline schedule is constructed by solving the well-known deterministic *resource-constrained project scheduling problem* (RCPSP). This problem (problem $(m, 1|cpm|C_{max})$ in the notation of Herroelen et al (2000)) involves the determination of a baseline schedule that satisfies both the finish-start, zero-lag precedence constraints between the activities and the renewable resource constraints under the objective of minimizing the project duration (for recent comprehensive overviews of the literature, we refer to Demeulemeester and Herroelen (2002) and Neumann et al (2003)).

A baseline schedule serves a number of important functions (Aytug et al (2005), Mehta & Uzsoy (1998), Wu et al (1993)). One of these is to provide internal visibility within the organization of the planned activity execution periods reflecting the requirements for the key staff, equipment and other resources. The baseline schedule is also the starting point for communication and coordination with external entities in the company's inbound and outbound supply chain: it constitutes the basis for agreements with suppliers and subcontractors (e.g. for planning external activities such as material procurement and preventive maintenance), as well as for commitments to customers (delivery dates).

During execution, however, a project may be subject to considerable uncertainty, which may lead to numerous schedule disruptions. Activities can take shorter or longer than primarily expected, resource requirements or availabilities may vary, ready times and due dates may change, new activities might have to be inserted in the schedule, etc. In this chapter, we limit ourselves to the treatment of *time uncertainties* caused by the fact that actually realized activity durations during project execution may deviate from the durations that were planned in the baseline schedule.

In general, there are two approaches to dealing with uncertainty in a scheduling environment (Davenport and Beck (2002), Herroelen and Leus (2005)): *pure reactive scheduling* and *proactive-reactive scheduling*.

Pure reactive scheduling does not rely on a predictive schedule; the use of schedules is eliminated altogether. Dynamic scheduling decisions are made during project execution at stochastic decision points, usually corresponding to the completion times of activities. The decisions are made by deploying so-called *scheduling policies* or *scheduling strategies* for determining which activities to dispatch throughout time. The policies rely on the observed past and the a priori knowledge of the distribution of the activity and resource characteristics. An extensive characterization of scheduling policies and subclasses can be found in Möhring et al (1984, 1985). We refer to Part 2 of Pinedo (2002), Chapter 9 in Demeulemeester and Herroelen (2002), Stork (2001) and Herroelen and Leus (2005) for a project scheduling setting.

Pure reactive scheduling lies outside the scope of this chapter. We focus on *proactive-reactive project scheduling*, i.e. the creation of a proactive baseline schedule that is as much as possible protected against disruptions and the deployment of reactive scheduling procedures during project execution to revise or reoptimize the schedule when necessary.

The remainder of this chapter is organized as follows. Section 2.2 focuses on the main principles of proactive-reactive scheduling and dwells on schedule robustness and its measurement. Section 2.3 focuses on the development of proactive project schedules. A number of recently developed proactive scheduling heuristics are described and their working principles are illustrated on a problem example. Section 2.4 describes and illustrates reactive procedures on the same problem example. Section 2.5 gives an overview of the experimental results on the makespan-stability trade-off.

2.2 Proactive-reactive project scheduling

Proactive-reactive scheduling involves a proactive and a reactive phase. During the *proactive* phase, a baseline schedule is constructed that accounts for statistical knowledge of uncertainty and anticipates disruptions. The underlying idea is to protect the schedule as well as possible from the disruptions that may take place during the execution of the project. When disruptions do occur during actual project execution, it may be necessary to call upon *reactive* scheduling procedures to modify the baseline schedule in response to these disruptions. The schedule that is obtained after these modifications is called the *realized schedule* (Aytug et al (2005)).

In general terms, a baseline schedule that is rather 'insensitive' to disruptions that may occur during project execution is called *robust.* The robustness concept has been used in many disciplines (see e.g. Kouvelis and Yu (1997), Roy (2002), Billaut et al (2005)). Many different types of robustness have been identified in the literature, calling for rigorous robustness definitions and the use of proper robustness measures.

2.2.1 Robustness types and measures

A robustness measure can be single or composite. Two often used types of *single robustness measures* have been distinguished: solution and quality robustness (Herroelen and Leus (2005); for other typologies we refer to Sanlaville (2004)).

2.2.1.1 Solution robustness or schedule stability. Solution robustness or schedule stability refers to the *difference* between the baseline schedule and the realized schedule. The difference or *distance* $\Delta(S, \mathrm{S})$ between the baseline schedule S and the realized schedule $\mathbf{S}$ for a given execution scenario can be measured in a number of ways: the number of disrupted activities, the difference between the planned and realized activity start times, etc.

For example, the difference can be measured by the weighted sum of the absolute deviation between the planned and realized activity start times, i.e.

$$\Delta(S, \mathrm{S}) = \sum_j w_j \left|\mathrm{s}_j - s_j\right|, \qquad [1]$$

where s_j denotes the planned starting time of activity j in the baseline schedule S, s_j is a random variable denoting the actual starting time of activity j in the realized schedule $\mathbf{S}$, and the weights w_j represent the disruption cost of activity j per time unit, i.e. the non-negative cost per unit time overrun or underrun on the start time of activity *j*. This cost reflects either the difficulty in shifting the booked time window on the required resources (internal stability, or the difficulty in obtaining the required resources) or the importance of on-time performance of the activity (external stability). The objective of the proactive-reactive scheduling procedure is then to minimize $\sum_j w_j E\left|\mathrm{s}_j - s_j\right|$, with E denoting the expectation operator, i.e. to minimize the weighted sum of the expected absolute difference between the planned and realized activity start times. It should be observed that the analytic evaluation of this objective is very cumbersome. For NP-hardness proofs of several cases of the scheduling problem for stability subject to a deadline and discrete disturbance scenario, we refer to Leus and Herroelen (2005).

Sanlaville (2004) suggests to measure solution robustness as

$$\max_I \Delta(S, \mathbf{S}), \qquad [2]$$

the maximum difference between the baseline schedule *S* and the realized schedule **S** over the set of execution scenarios *I*. The objective of the proactive-reactive scheduling procedure then is to minimize this maximum distance.

2.2.1.2 Quality robustness. Quality robustness refers to the insensitivity of some deterministic objective value of the baseline schedule to distortions. The ultimate objective of a proactive-reactive scheduling procedure is to construct a baseline schedule for which the objective function value does not deteriorate when disruptions occur. The quality robustness is measured in terms of the value of some objective function z. In a project setting, commonly used objective functions are project duration (makespan), project earliness and tardiness, project cost, net present value, etc.

When stochastic data are available, quality robustness can be measured by considering the *expected value of the objective function*, such as the expected makespan $E\left[C_{\max}\right]$, the classical objective function used in stochastic resource-constrained project scheduling (Stork (2001)).

It is logical to use the *service level* as a quality robustness measure, i.e. to maximize $P(\mathrm{z} \leq z)$, the probability that the objective function value of the realized schedule stays within a certain treshold z. For the makespan objective, we want to maximize the probability that the project completion time does not exceed the project due date δ_n, i.e. $P(\mathrm{s}_n \leq \delta_n)$, where s_n denotes the starting time of the dummy end activity. Van de Vonder et al (2005a) refer to this measure as the *timely project completion probability* (TPCP). It should be observed that even the analytic evaluation of this measure for a given schedule and in the presence of ample resource availability is very troublesome, the PERT problem being #P-complete (Hagstrom (1988)).

Quality robustness can also be measured by comparing the solution value z of the realized schedule obtained by the proactive-reactive scheduling procedure and the optimal solution value z^* computed ex-post by applying an exact procedure on the basis of the realized activity durations. Leus and Herroelen (2001), for example, have used the percentage deviation of $C_{\max}$, the project duration of the realized schedule, from the ex-post optimal makespan $C^*_{\max}$ computed by applying a branch-and-bound procedure on the basis of the actually realized activity durations, as a measure of quality robustness.

2.2.1.3 Composite robustness measures. The robustness measures described above are all single measures. It is also possible to use composite objectives (Hoogeveen (2005)). Van de Vonder et al (2005b) use the bi-criteria objective $F(P(\mathrm{s}_n \leq \delta_n), \sum w_j E\,|\mathrm{s}_j - s_j|)$ of maximizing the timely project completion probability and minimizing the weighted sum of the expected absolute deviation in activity starting times. The authors assume that the composite objective function $F(.,.)$ is not known a priori and that the relative importance of the two criteria is not known in the initial schedule development phase, i.e. the decision maker has no knowledge of e.g. a linear combination that reflects his preference. Analytic evaluation of a composite objective function is usually very cumbersome (as mentioned before, the PERT problem is $\#P$-complete (Hagstrom (1988)) and the scheduling problem for stability is NP-hard in the ordinary sense (Leus and Herroelen (2005))). A natural way out is to evaluate the composite objective function by means of simulation.

2.3 Proactive project scheduling procedures

The development of proactive project scheduling procedures is still in its burn-in phase. A common characteristic of the proactive project baseline scheduling procedures in the sparse open literature is the insertion of *time buffers* in the schedule (Herroelen and Leus (2005)). For the alternative approach of generating multiple schedules (contingent scheduling), we refer to Billaut and Roubellat (1996ab) and Artigues et al (1999, 2005). *Contingent scheduling* is based on the generation of multiple baseline schedules (or baseline schedule fragments) before and/or during project execution that respond to foreseen disruptive events, or are equivalent in performance. Responding to anticipated events during schedule execution is then simply done by switching to the schedule (fragment) that corresponds to the events that have occurred. The approach focuses on *flexibility* rather than robustness, and is especially designed for time-critical reactive scheduling.

2.3.1 Critical chain scheduling

The *critical chain* methodology (Goldratt (1997)) has resulted in a number of commercial software packages (ProChain® (www.prochain.com), cc-Pulse® (www.sphericalangle.com), CCPM+® (www.advanced-projects.com), PS Suite® (www.sciforma.com)) allowing to generate buffered baseline schedules. The idea is to remove the safety that is normally included in the individual activity duration estimates by using aggressive estimates based on the mean or median duration and to concentrate the safety into project and feeding buffers. The critical chain (*CC*) is defined on a precedence and resource feasible schedule as the chain of precedence and/or resource dependent activities that determines the project duration. If there is more than one candidate critical chain, an arbitrary one is chosen. A project buffer is inserted at the end of the *CC* to protect the project due date against variation in the *CC*. Feeding buffers are inserted wherever non-critical chains meet the *CC* in order to prevent distortions in the non-critical chains from propagating throughout the *CC*. The default buffer size is fifty percent of the length of the chain feeding the buffer. Alternative buffer sizing procedures have been presented in the literature (Newbold (1998), Tukel et al (2005)).

The potentials and pitfalls of the *CC*-methodology have been discussed by Herroelen and Leus (2001), Elmaghraby et al (2003) and Herroelen et al (2002). The main conclusion that can be drawn from these studies is that the project buffer may overprotect the project makespan and may lead to unnecessarily high project due dates, while the feeding buffers may fail to prevent propagation of schedule disruptions throughout the baseline schedule.

2.3.2 Generating stable project schedules with ample resource availability

Herroelen and Leus (2004) develop mathematical programming models for the generation of stable baseline schedules in a project environment. The authors make abstraction of resource usage, assuming that a proper allocation of resources has been performed. They use the concept of *pair-wise float*, defined as the difference between the start time of activity *j* and the finish time of activity *i* in a schedule S. The pair-wise float is only defined for activities $(i, j) \in TA$, where TA denotes the transitive closure of A, meaning that $(i, j) \in TA$ if and only if a path from i to j exists in the activity-on-the-node project network $G = (N, A)$. The authors select a project due date δ_n and assign a probability of disruption p_j to every activity j (j=0,1,. . . ,n), with $\sum_{j=0}^{n} p_j = 1$. The dummy end node has disruption probability $p_n = 0$, while p_0 denotes the probability that the dummy start node, i.e. the entire project, starts later than initially anticipated. They use a random variable L_j to denote the disturbance length of activity j if it is disturbed, and a non-negative cost w_j per unit time overrun on the start time of activity *j*.

The authors propose to use the expected weighted deviation in start times in the realized schedule from those in the pre-schedule as stability measure. They set up a linear program that can be rewritten as the dual of a minimum cost network flow problem and report on very promising computational results obtained on a set of randomly generated test instances.

Van de Vonder et al (2005a) investigate the potential trade-off between quality robustness (measured in terms of the timely project completion probability $P(\mathrm{s}_n \leq \delta_n)$) and solution robustness, measured by $\sum_{j=0}^{n} w_j E|\mathrm{s}_j - s_j|$. Using simulation, they analyze the quality and solution robustness of critical chain-based schedules discussed above (as representatives of makespan protecting schedules) and the *activity dependent float factor model* (ADFF), shown by Leus (2003) to produce good solution robustness results when the number of disruptions (activity duration increases) is rather high. The procedure is an adaptation of the float factor model that was originally introduced by Tavares et al (1998) to generate a schedule S in which the start time of activity j is obtained as $s_j(S) := s_j(ESS) + \alpha_j(s_j(LSS) - s_j(ESS))$, where $\alpha_j \in [0,1]$ is the so-called float factor, $s_j(ESS)$ denotes the earliest possible start time of activity j and $s_j(LSS)$ represents the latest allowable start time of activity j. Both start times are derived from critical path calculations for a given project deadline. Instead of using a single float factor α for all the activities, ADFF adopts an activity dependent float factor that is calculated as $\alpha_j = \beta_j/(\beta_j + \lambda_j)$, where β_j is the sum of the weight of activity j and the weights of all transitive predecessors of activity j, while λ_j is the sum of the weights of all transitive successors of activity j. In doing so, ADFF inserts longer time buffers in front of activities that would incur a high cost if started later than originally planned.

The conclusions reached by the authors are counterintuitive. While the pioneers of the critical chain methodology focus on due date performance and the critical chain scheduling procedures basically aim at due date protection, the critical chain scheduling approach was found to be outperformed by ADFF on the instances where the timely realization of the project (expressed in terms of a large w_n) is deemed important.

2.3.3 Restricted resources and robust resource allocation

If the unrestricted resource availability assumption is dropped from the analysis, Leus and Herroelen (2004) use a so-called *resource flow network* to represent the flow of resources across the activities of the project network and study the problem of generating a robust resource allocation under the assumption that a feasible baseline schedule exists and that some advance knowledge about the probability distribution of the activity durations is available. The authors explore the fact that checking the feasibility of a resource allocation can easily be done using maximal flow computations in the resource flow network. As such, the search for an optimal allocation is reduced to the search for an associated resource flow network with desirable robustness characteristics. The authors propose a branch-and-bound algorithm that solves the *NP*-hard robust

Table 2.1. Data for the problem instance

j	$E(\mathbf{d}_j)$	w_j	r_j	var	dis
0	0	0	0	none	+0
1	4	2	5	medium	+3
2	5	1	3	low	+0
3	2	3	4	medium	+0
4	4	1	4	high	+1
5	5	8	3	medium	−2
6	4	1	5	low	−1
7	2	3	3	high	+2
8	2	6	6	low	+1
9	0	30	0	none	+0

resource allocation problem (Leus (2003)) in exact and approximate formulations. The procedure heavily relies on constraint propagation during its search. The authors report on promising results obtained on a set of problem instances that were generated using the problem generator RanGen (Demeulemeester et al (2003)).

2.3.4 Proactive scheduling procedures

In order to illustrate the working principles of a number of recently developed proactive project scheduling procedures, we use the well-known RCPSP as our vehicle of analysis. The activity-on-the-node project instance of Figure 2.1 represents 10 activities (activity 0 and activity 9 are dummy activities representing the project start and finish) subject to finish-start, zero-lag precedence constraints and a single renewable resource constraint. The availability of the single renewable resource type is 10 units per period.

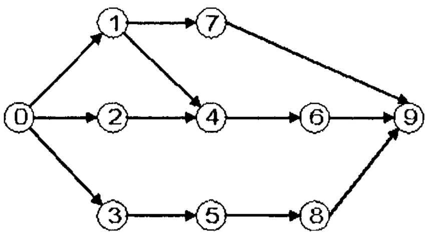


Figure 2.1. Problem Instance

The data for the problem instance are shown in Table 2.1. The first column shows the activity numbers. The mean duration E($\mathbf{d}_j$) of each activity is shown in the second column. The activity weights w_j are shown in the third column. The next column contains the per period requirements r_j of the single renewable resource type. The column labelled *var* gives an indication of the activity duration variability. *High* duration variability means that the real activity durations are discretized values drawn from a right-skewed beta distribution with parameters 2 and 5, that is transformed in such a way that the minimum duration equals 0.25 the expected duration, the mean equals the expected duration and the maximum duration equals 2.875 times the expected duration. *Low* duration variability means that the realized activity durations are also discretized values drawn from a beta distribution with parameters 2 and 5, but with the mean equal to the expected activity duration and with minimal and maximal values equal to 0.75 times and 1.625 times the expected activity duration, respectively. *Medium* duration variability means that the realized activity durations are drawn from a beta distribution with parameters 2 and 5, but with minimum and maximal values equal to 0.5 times and 2.25 times the expected activity duration, respectively. Obviously, *none* refers to a deterministic activity duration. The column labelled *dis* denotes the disturbance in the activity duration that is assumed to occur during the realization of the project. The actual duration of activity 1, for example, will exceed its planned duration by 3 time periods, while the actual duration of activity 5 will be 2 time periods smaller than its planned duration.

2.3.4.1 An exact procedure for generating quality robust baseline schedules. Figure 2.2 shows the minimum makespan schedule obtained by the branch-and-bound procedure of Demeulemeester and Herroelen (1992, 1997) for the RCPSP defined on the problem instance of Figure 2.1, using the expected activity durations given in Table 2.1. The project duration equals $C_{\max} = 15$. This schedule can be used as the baseline. Assume that the project due date is set 30% above this minimum makespan, i.e. $\delta_9 = \lceil 1.3 \times C_{\max} \rceil = 20$. The five-period time interval between the minimum project completion time and the due date acts as a protective time cushion during project execution, inducing quality robustness into the schedule.

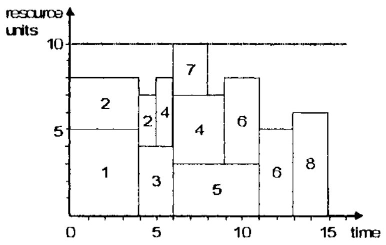


Figure 2.2. Minimum makespan schedule

2.3.4.2 A suboptimal procedure for generating quality robust baseline schedules. The RCPSP is strongly *NP*-hard (Blazewicz et al (1983)) so that solving it to optimality may entail a large computational effort. That explains why commercial software packages generally rely on simple priority-based scheduling heuristics for generating a baseline schedule. The experimental study by Kolisch (1996) revealed that the well-known *Late Start Time* (LST) priority rule ranks among the best priority-rule based procedures.

Relying on the critical path based latest allowable start times for the example project of Figure 2.1, the LST-heuristic creates the priority list *L*=(0, 2,1,3,4,5,6,7,8,9) with the activities ranked in increasing order of their latest allowable start time. Applying a serial schedule generation scheme yields by lucky coincidence the minimum duration schedule of Figure 2.2.

2.3.4.3 Suboptimal procedures for generating solution robust baseline schedules. Solution robust scheduling procedures insert time buffers in the baseline schedule to absorb expected distortions. In the remainder of this section, we discuss and illustrate several procedures for inserting time buffers into the minimum duration schedule of Figure 2.2. During project execution, activities will never be allowed to start earlier than planned in order to preserve the stability advantage of the idle times in the schedule. This execution policy is commonly referred to as railway scheduling, because of its comparability with the scheduling of trains in a railway station.

The resource flow dependent float factor heuristic (RFDFF). The *resource flow dependent float factor* (RFDFF) heuristic has been developed by Van de Vonder et al (2006) as an extension of the *activity dependent float factor* (ADFF) heuristic mentioned earlier. This heuristic completely relies on the activity weights, but does not exploit the available information offered by the activity duration distributions in making its buffering decisions.

The starting time of activity j in the RFDFF schedule is calculated as $s_j(S) := s_j(B\&B) + \alpha_j(float(j))$, where $s_j(B\&B)$ denotes the starting time of activity j in the minimum duration baseline schedule and α_j now denotes the *resource flow dependent float factor.* The total float, $float(j)$, is the difference between the latest allowable starting time of activity j given the project due date (i.e. its starting time in the right-justified version of the minimum duration schedule) and its scheduled starting time in the minimum duration schedule.

To calculate the float factors α_j, we first need to construct a *resource flow network* (Artigues and Roubellat (2000), Leus (2003)) for the minimum duration schedule. The resource flow network is a network with the same nodes as the original project network, but with arcs connecting two nodes if there is a resource flow between the corresponding activities. It thus identifies how each single item of a resource is passed on between the activities in the schedule. A schedule may allow for different ways of allocating the resources so that the same schedule may give rise to different resource flow networks. We use the (one-pass) algorithm of Artigues & Roubellat (2000) to construct a resource flow network. We have redrawn in Figure 2.3 the minimum makespan schedule of Figure 2.2 to illustrate the resulting use of the individual resource units along the horizontal bands. Figure 2.4 shows the corresponding resource flow

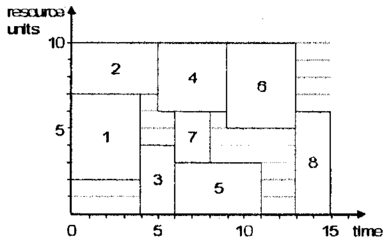


Figure 2.3. Minimum makespan schedule

network. Activity 8, for example, has a per period resource requirement of six units. It uses three resource units released by its predecessor activity 5, two units passed on by activity 7, and one unit released by activity 6. The float fac-

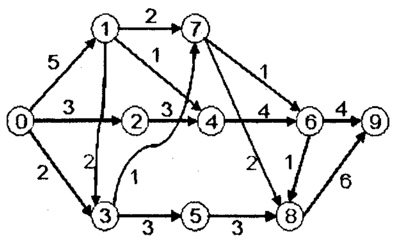


Figure 2.4. Resource flow network

tors α_j are again calculated as $\alpha_j = \beta_j / (\beta_j + \lambda_j)$, where β_j now is the sum of the weight of activity j and the weights of all its transitive predecessors in both the original network and the resource flow network, while λ_j is the sum of the weights of all transitive successors of activity j in both networks. The weights of activities that start at time 0 are not included in these summations because it is assumed that these activities can always start at their planned start time and thus do not need any buffering to cope with possible disruptions of their predecessors. The RFDFF heuristic consequently inserts longer time buffers in front of activities that would incur a high cost if started earlier or later than originally planned and resource constraints will always remain satisfied in the resulting schedule.

Table 2.2. Values for the RFDFF heuristic

j	$s_j(B\&B)$	$float(j)$	w_j	β_j	λ_j	α_j	s_j
0	0	5	0	0	52	0	0
1	0	6	0	0	52	0	0
2	0	5	0	0	38	0	0
3	4	6	3	3	48	0.059	4
4	5	5	1	1	37	0.026	5
5	6	7	8	11	36	0.234	8
6	9	5	1	8	36	0.182	10
7	6	6	3	6	37	0.140	7
8	13	5	6	22	30	0.423	15
9	20	5	30	52	0	1	20

Table 2.2 lists for each activity of the example project the values needed for the computation of the resource flow dependent float factor α_j and the scheduled starting time s_j. Note that we set $w_1 = w_2 = 0$ because activities 1 and 2 start at time 0. Activity 6 has activity 4 as direct predecessor in the project network of Figure 2.1 and activities 4 and 7 as direct predecessors in the resource flow network of Figure 2.4. Activities 0, 1, 2 and 3 are its transitive predecessors in these networks. This results in $\beta_6 = w_6 + w_3 + w_4 + w_7 = 8$. Similarly, summing the weights of all direct and transitive successors of activity 6 in the networks of Figure 2.1 and Figure 2.4 gives $\lambda_6 = w_8 + w_9 = 36$. We thus find $\alpha_6 = 8/(36+8) = 0.182$ and $s_6 = 9 + 0.182 \times 5 = 9.91$, which is discretized (rounded to the nearest integer) to 10. The starting times of the other activities can be calculated in the same way, resulting in the RFDFF schedule of Figure 2.5.

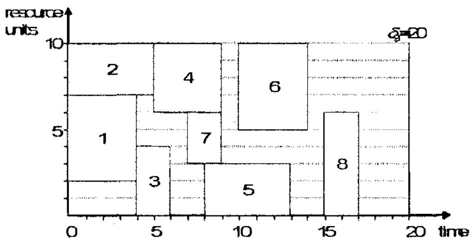


Figure 2.5. RFDFF schedule

The virtual activity duration extension heuristic (VADE). The *virtual activity duration extension* (VADE) heuristic, developed by Van de Vonder et al (2005c), starts from a different point of view. The standard deviations σ_j of the activity durations, assumed known, are used to iteratively compute virtual duration extensions for the non-dummy activities. These virtual activity durations are used to update the activity start times and, by doing so, insert time gaps in the baseline schedule. The updated activity starting times are then used to generate the buffered baseline schedule using the original expected activity durations.

The iterative procedure works as follows:

```
For $j = 1, 2, ..., n-1$ do $d_j^* = E(\mathbf{d}_j)$ and $v_j = 1$;
Compute $s_j$, $j = 1, 2, ..., n$;
While $s_n \leq \delta_n$ do
        Find $j^* : \frac{v_j^*}{\sigma_{j^*}} = \min_j \left\{\frac{v_j}{\sigma_j}\right\}$
        (tie-break: $\max sw_i = \sum_{i=succ(j)} w_i$);
        $v_j^* = v_j^* + 1$;
        $d_j^* = d_j^* + 1$;
        Compute $s_n$;
Generate the buffered baseline schedule.
```

Initially each non-dummy activity duration d_j^*, $j = 1, 2, ..., n-1$ is set equal to its expected value $E(\mathbf{d}_j)$ and all $v_j = 1$. The initial activity start times s_j, $j = 1, 2, ..., n$, are computed by creating an early start schedule for the resource flow network using the activity durations d_j^*. As long as the project duration stays within the due date, the activity start times are iteratively updated as follows. Determine the activity j^* for which $\frac{v_j^*}{\sigma_{j^*}} = \min_j \left\{\frac{v_j}{\sigma_j}\right\}$. Ties are broken by selecting the activity for which the sum of the weights of all its non-dummy successors is the smallest. Set $v_j^* = v_j^* + 1$ and $d_j^* = d_j^* + 1$. If necessary, update the schedule and reiterate.

The standard deviations of the activity durations of our project example are calculated as shown in Table 2.3. As $\frac{v_4^*}{\sigma_{4^*}} = 1.19 = \min_j \left\{\frac{v_j}{\sigma_j}\right\}$, $d_4^* = 4+1 = 5$ and $v_4^* = 1+1 = 2$, so that $\frac{v_4}{\sigma_4} = 2.38$. The virtual duration extension of activity 4 generates a one-period delay in the starting times of its successor activities 6 and 8 in the resource flow network, so that $s_6 = 10$ and $s_8 = 14$. Next, $\frac{v_5^*}{\sigma_{5^*}} = 1.59 = \min_j \left\{\frac{v_j}{\sigma_j}\right\}$ so that $d_5^* = 5+1 = 6$, $v_5^* = 1+1 = 2$ and $\frac{v_5}{\sigma_5} = 3.17$. The current schedule does not need to be modified because of the two-period gap between the completion time of activity 5 and the starting time of activity 8. Now we have that $\frac{v_7^*}{\sigma_{7^*}} = 1.69 = \min_j \left\{\frac{v_j}{\sigma_j}\right\}$, $d_7^* = 2+1 = 3$ and $v_7^* = 1+1 = 2$, so that $\frac{v_7}{\sigma_7} = 3.39$. Activity 7 has a two-period float, so there is no need to update the schedule. Now $\frac{v_1^*}{\sigma_{1^*}} = 1.79 = \min_j \left\{\frac{v_j}{\sigma_j}\right\}$, $d_1^* = 5$, $v_1^* = 2$ and $\frac{v_1}{\sigma_1} = 3.57$. Continuing the procedure in this manner leads to the virtual duration updates $d_4^* = 6$, $d_3^* = 3$, $d_5^* = 7$, $d_2^* = 6$, and $d_7^* = 4$. Now we

Table 2.3. Values for the VADE heuristic

j	$E(\mathbf{d}_j)$	σ_j	sw_i
1	4	0.56	7
2	5	0.31	1
3	2	0.40	11
4	4	0.84	1
5	5	0.63	6
6	4	0.28	6
7	2	0.59	7
8	2	0.20	0

have $\frac{v_1^*}{\sigma_{1^*}} = \frac{v_4^*}{\sigma_{4^*}} = \frac{v_6^*}{\sigma_{6^*}} = 3.57 = \min_j \left\{ \frac{v_j}{\sigma_j} \right\}$. Invoking the tie-break rule (see column sw_i in Table 2.3) leads to the update $d_1^* = 6$. Subsequently, $d_6^* = 5$, $d_4^* = 7$ and $d_5^* = 8$. At this juncture, updating $d_4^* = 8$ would move the expected project duration beyond the project due date. The algorithm terminates. The use of the updated start times and the original activity durations $E(\mathbf{d}_j)$ yield the buffered baseline schedule of Figure 2.6. Observe that the planned project completion time $\delta_9 = 20$ is not directly protected. Because activity 8 has low variability, the procedure prefers to protect its starting time rather than protecting the project completion time against disruptions of activity 8.

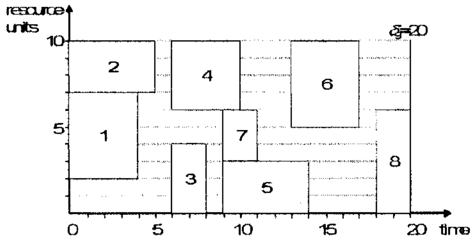


Figure 2.6. VADE schedule

The starting time criticality (STC) heuristic. The *starting time criticality heuristic* (STC) exploits information about both the weights of the activities and the variance structure of the activity durations (Van de Vonder et al. (2005c)). The basic idea is to start from a minimum duration schedule and iteratively create intermediate schedules by inserting a one-time period buffer in front of the activity that is the most starting time critical in the current intermediate schedule, until adding more safety would no longer improve stability. The starting time criticality of an activity j is defined as $stc(j) = P(\mathrm{s}(j) > s(j)) \times w_j = \gamma_j \times w_j$, where γ_j denotes the probability that activity j cannot be started at its scheduled starting time.

The iterative procedure runs as follows. At each iteration step (see pseudocode below) the buffer sizes of the current intermediate schedule are updated as follows. The activities are listed in decreasing order of the $stc(j)$. The list is scanned and the size of the buffer to be placed in front of the currently selected activity from the list is augmented by one time period and the starting times of the direct and transitive successors of the activity are updated. If this new schedule has a feasible project completion ($s_n < \delta_n$) and results in a lower estimated stability cost ($\sum_{j \in N} stc(j)$), the schedule serves as the input schedule for the next iteration step. If not, the next activity in the list is considered. Whenever we reach an activity j for which $stc(j) = 0$ (all activities j with $s_j = 0$ are by definition in this case) and no feasible improvement is found, a local optimum is obtained and the procedure terminates.

```
Iteration step
    Calculate all $stc(j)$
    Sort activities by decreasing $stc(j)$
    While no improvement found do
        take next activity $j$ from list
        if $stc(j)=0$: procedure terminates
        else add buffer in front of $j$
            update schedule
            if improvement & feasible do
                store schedule
                goto next iteration step
            else
                remove buffer in front of $j$
                restore schedule
```
The iteration step of the STC heuristic

Regrettably, the probabilities γ_j are not easy to compute. We define $k(i,j)$ as the event that predecessor i disturbs the planned starting time of activity j. The probability that this event occurs can be expressed as $P(k(i,j)) = P(\mathrm{s}_i + \mathrm{d}_i + LPL(i,j) > s_j)$ in which $LPL(i,j)$ is the length of the longest path between activity i and activity j in the resource flow network defined on the minimum duration schedule. γ_j can then be calculated as $\gamma_j = P(\bigcup_{i \in X} k(i,j))$, with X being defined as the set of all direct and transitive predecessors of j in the original network and the resource flow network. STC makes two assumptions in approximating γ_i: (a) predecessor i of activity j starts at its originally planned starting time when calculating $k(i,j)$ and (b) only one activity at a time disturbs the starting time of activity j. Assumption (b) means that we estimate $P(\bigcup_{i \in X} k(i,j))$

Table 2.4. The longest path lengths from i to j

$LPL(i,j)$	1	2	3	4	5	6	7	8	9
1			0	0	2	4	2	8	10
2				0		4		8	10
3					0	2	0	6	8
4						0		4	6
5								0	2
6								0	2
7						0		4	6
8									0

by $\sum_{i \in X} P(k(i,j))$, i.e. we assume that $P(k(i1,j) \cap k(i2,j)) = 0$ for each $i1, i2 \in X$. Assumption (a) boils down to setting $\mathbf{s}_i = s_i$. Combining both assumptions yields $\gamma'_j = \sum_{i \in X} P(\mathbf{d}_i > s_j - s_i - LPL(i,j))$ such that $stc(j) = \gamma'_j \times w_j$. Because s_i, s_j and $LPL(i,j)$ and the distribution of $\mathbf{d}_i$ are all known, we can now easily calculate all values of γ'_j and $stc(j)$ for every activity j.

The application of STC to the problem example runs as follows. First, the $LPL(i,j)$ values need to be calculated for all predecessors i for every activity j. For illustrative purposes, we calculate $LPL(1,3)$, $LPL(1,5)$ and $LPL(1,8)$. A glance at the minimum duration schedule of Figure 2.2 and the resource flow network of Figure 2.4 reveals that activity 3 is immediately preceded by activity 1 and thus $LPL(1,3) = 0$. The resource flow network of Figure 2.4 shows a unique path $< 1,3,5 >$ leading from activity 1 to activity 5. This results in $LPL(1,5) = E(\mathbf{d}_3) = 2$. Multiple paths exist between activity 1 and 8, namely $< 1,3,5,8 >$, $< 1,3,7,8 >$, $< 1,3,7,6,8 >$, $< 1,7,8 >$, $< 1,7,6,8 >$ and $< 1,4,6,8 >$ with a corresponding path length of 7, 4, 8, 4, 2, 6 and 8, respectively. Thus, $LPL(1,8) = 8$. Table 2.4 shows all $LPL(i,j)$-values. If i is no transitive predecessor of j in either the original network or the resource flow network, the corresponding cell in the table is left blank.

The *stc*-values are first calculated for the initial minimum duration schedule of Figure 2.2 with $s_n = \delta_n = 20$. For example $stc(6)$ is calculated as $w_6 \times (k(1,6) + k(2,6) + k(3,6) + k(4,6) + k(7,6))$ with
$k(1,6) = P(\mathbf{d}_1 > s_6 - s_1 - LPL(1,6)) = P(\mathbf{d}_1 > 9 - 0 - 4) = P(\mathbf{d}_1 > 5) = 0.11$
$k(2,6) = P(\mathbf{d}_2 > s_6 - s_2 - LPL(2,6)) = P(\mathbf{d}_2 > 5) = 0.23$
$k(3,6) = P(\mathbf{d}_3 > s_6 - s_3 - LPL(3,6)) = P(\mathbf{d}_3 > 3) = 0.01$
$k(4,6) = P(\mathbf{d}_4 > s_6 - s_4 - LPL(4,6)) = P(\mathbf{d}_4 > 4) = 0.34$

$k(7,6) = P(\mathbf{d}_7 > s_6 - s_7 - LPL(7,6)) = P(\mathbf{d}_7 > 3) = 0.05$
This results in $stc(6) = 1 \times (0.11 + 0.23 + 0.01 + 0.34 + 0.05) = 0.74$. All the *stc*-values for the starting solution are shown in column *Initial* in Table 2.5. $\sum stc(i) = 13.25$ denotes the total cost of the schedule and provides a good measure for the stability of the schedule. Ordering the activities by decreasing *stc* gives (**8**, 5, 7, 3, 6, 4, 9, 1, 2). Adding a one-time period buffer in front of activity 8 yields the feasible starting time $s_8 = 14$ and provides the input schedule for Step 1.

The newly inserted buffer in front of activity 8 requires a recalculation of its *stc*-value and the *stc*-value of its successor activity 9. However, observe that $stc(9)$ does not change although the buffer between activities 8 and 9 is reduced in size by one time period. Indeed, activity 8 has a very low variability and $P(k(8,9)) = 0$, indicating that the buffer size between activity 8 and 9 does not need to be increased. Table 2.5 shows the new values in the Step 1 column. Activity 5 now has the largest *stc*-value, yielding the ordered list (**5**, 8, 7, 3, 6, 4, 9, 1, 2). Delaying the starting time of activity 5 by one time period is feasible and leads to a reduction in the total schedule cost.

In Step 2, $stc(5)$ will decrease while $stc(8)$ will increase because of the delay of s_5. Activity 8 has the largest *stc*-value and will be delayed by one time period in the input schedule of Step 3. The buffer size in front of activity 8 now equals two time periods.

Similar to Step 1, the buffer in front of activity 9 is large enough and thus only $stc(8)$ drops in value, resulting in the new ordered list (**7**, 5, 3, 6, 8, 4, 9, 1, 2). Inserting a one-time period buffer in front of activity 7 will not cause any feasibility problems and will improve the total schedule stability cost.

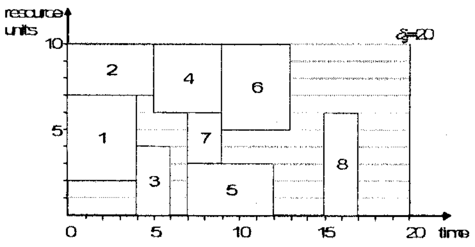


Figure 2.7. The input schedule for step 4

The input schedule for Step 4 is given in Figure 2.7. Obviously, the insertion of the buffer in front of activity 7 will affect $P(k(7,6))$, the probability that the highly variable activity 7 delays the starting time of activity 6, which will result in an increase of $stc(6)$. Also $stc(8)$ will slightly increase. It is no surprise that activity 6 becomes the most time critical activity and will thus be delayed.

For the input schedule of Step 5, we have that $s_6 = 10$. However, because the buffer size in front of activity 8 was previously fixed at two time periods,

Table 2.5. Computational steps of the STC procedure

Act	*Initial*	*Step* 1	*Step* 2	*Step* 3	*Step 4*	Step 5	Step 6	Step 7
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.33
4	0.34	0.34	0.34	0.34	0.34	0.34	0.34	0.34
5	3.91	3.91	0.93	0.93	0.93	0.93	0.18	0.02
6	0.74	0.74	0.74	0.74	0.94	0.29	0.29	0.14
7	1.47	1.47	1.47	1.47	0.35	0.35	0.35	0.09
8	5.86	1.57	1.79	0.58	0.60	0.20	0.26	0.11
9	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.06
Tot	13.25	8.96	6.2	4.99	4.09	3.04	2.35	1.09

also s_8 will be delayed by one time period. This results in the positive side effect that also $stc(8)$ will decrease. The ordered list becomes (**5**, 3, 7, 4, 6, 8, 9, 1, 2) and the total schedule cost has decreased in value to 3.04.

In Step 6 the buffer size in front of activity 5 will be set to two time periods. Alongside the positive impact on $stc(5)$ itself, this will have a slightly negative impact on $stc(8)$ due to a larger value of $P(k(5, 8))$. Observe the difference with the previous step. Here the buffer size in front of activity 8 remains respected by delaying a predecessor, while in Step 5, s_8 had to be delayed in order to respect the two-time period buffer with an inverse impact on $stc(8)$ as a result. Table 2.5 again shows the updated *stc*-values. Activity 3 is now the most time critical and will be delayed.

Inserting a buffer in front of activity 3 results in a major shift in the schedule. The starting times of activities 5, 6, 7 and 8 will all be delayed by one time period in order to keep their previously allocated buffer sizes fixed. All the corresponding *stc*-values will decrease and $\sum stc(i) = 1.09$. A small increase of $stc(9)$ is the only drawback. The *Step 7* column in Table 2.5 shows the new ordered list (4, 3, 6, 8, 7, 9, 5, 1, 2). The algorithm continues by trying to delay activity 4. By doing so $stc(4)$ would decrease from 0.342 to 0.050, but $stc(6)$, $stc(8)$ and $stc(9)$ would increase by a combined 0.436 resulting in a higher total schedule cost. Delaying activity 4 is thus infeasible and we proceed with the next activity in the list, namely activity 3. However, activity 3 and all other activities in the list can also be proven to be non-improving moves. The procedure terminates and Figure 2.8 represents the schedule found by the STC heuristic for our example project.

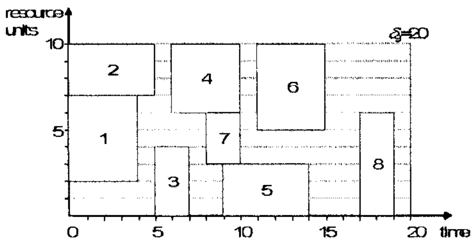


Figure 2.8. STC schedule

Tabu search. Van de Vonder et al (2005c) also introduce a tabu search procedure (Glover (1989, 1990)) for solution robust buffer allocation. The tabu search procedure departs from the VADE-schedule described in Section 2.3.4.3. At each iteration step, the neighborhood of the current solution contains at most $2 \times (n - 2)$ solutions. For each non-dummy activity of the project, we have two possible neighborhood solutions. One is obtained by increasing the buffer in front of the activity in the schedule by one time period, if possible (plus-move). The other is obtained by decreasing the buffer size of this activity by one unit, if possible (minus-move). The buffers in front of all other activities are left unchanged. For every neighborhood solution the expected stability cost will be estimated by simulation. Two tabu lists are kept, both of length $n/3$. The first list stores all recent plus-moves, while the second one stores all recent minus-moves. Before allowing a new plus-move, we have to check whether this activity is not in the second list. If a buffer size decrease (minus-move) delivers the best solution in the neighborhood, the first tabu list has to be checked. The best non-tabu move is selected and executed. By doing so, we avoid cycling, but we do allow an activity to be consecutively selected more than once if the considered moves have the same direction. The aspiration criterion defines that a move that would yield a new best solution will be accepted even if it would normally be prohibited by the tabu list. The overall best found solution is stored throughout the whole procedure. The tabu search stops after a fixed number of iterations.

Contrary to all previous heuristics, the outcome of the procedure is dependent on the simulated disruptions. In general, the tabu search will lead to more robust schedules than the ones obtained by the other heuristics, but suffers from a higher computational cost. An extensive simulation-based comparison of the results can be found in Van de Vonder et al (2005c).

2.4 Reactive scheduling procedures

Proactive-reactive scheduling implies that the buffered baseline schedules generated by the proactive procedures discussed above should be combined with reactive procedures that are deployed during project execution when disruptions occur that cannot be absorbed by the buffered schedule. Van de Vonder et al (2005b) have recently developed a number of reactive project scheduling procedures. We will illustrate them using the disruption scheme shown in the last column of Table 2.1 to be applied to the unbuffered minimum duration schedule of Figure 2.3.

2.4.1 The fix flow procedure

The fix flow procedure applies an early start policy at the schedule breakage point, while maintaining the resource allocation decisions made in the baseline schedule, as reflected in its resource flow network.

The disruption scenario shown in Table 2.1 reveals that activity 2 is not disturbed while activity 1 suffers from a duration increase of three time periods, changing its planned finish time from time instant 4 to time instant 7. At time instant 5, activity 2 is finished while activity 1 still has a remaining duration of two periods. Activity 4 is a direct successor of activity 1, and is now planned to start at time instant 7. Maintaining the resource flow of two resource units from activity 1 to 3 (see Figure 2.4) results in the fact that activity 3 is also delayed up to time instant 7. Planning the remaining activities as early as possible for the given resource flows yields the *projected schedule* of Figure 2.9. Such a projected schedule describes how the project will execute if no further disruptions occur, given all available information at the current time instant 5. Taking into account all the disruptions of the disruption scenario

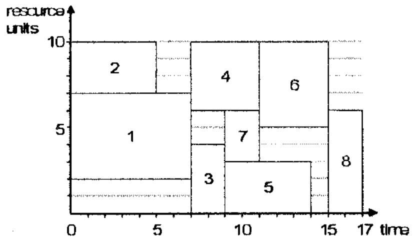


Figure 2.9. Projected schedule generated by the fix flow reactive procedure at time instant 5

shown in Table 2.1, the final 19-period realized schedule generated by the fix flow reactive procedure is shown in Figure 2.10. Denoting the realized and scheduled start times of activity *j* by s_j and s_j, respectively, the stability cost can be computed as $\sum w_j|\mathrm{s}_j - s_j| = 66$.

2.4.2 The weighted earliness-tardiness procedure

The *weighted earliness-tardiness* (WET) reactive procedure sets the due date for an activity equal to its projected finish time in the baseline schedule. The

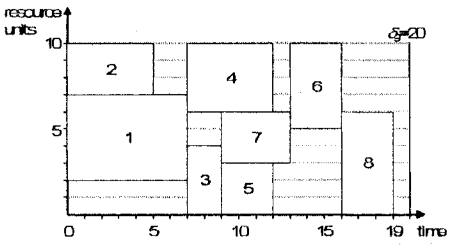


Figure 2.10. Fix flow reactive schedule

due date for the dummy end activity is set to the project due date δ_n. The unit earliness and tardiness costs of a non-dummy activity j are assumed to be identical and are set equal to its weight w_j. The earliness cost of the dummy end activity is set to zero, because we do not punish the project for finishing earlier than planned. The tardiness cost for the dummy end activity n is set to w_n. Upon schedule breakage we invoke the branch-and-bound algorithm developed by Vanhoucke et al (2001) for the resource-constrained earliness-tardiness project scheduling problem $(m, 1|cpm|early/tardy)$ in the notation of Herroelen et al (2000)). The idea is to construct at each rescheduling point a new projected schedule with minimum stability cost.

The WET procedure does not keep the resource flows fixed. The application of the procedure to the problem example and the disruption scenario of Table 2.1 changes the resource flow at time instant 5 when activity 2 finishes (see Figure 2.11). Since activity 1 suffers from a three-period duration increase, activity 3 will no longer wait for activity 1 to finish, but will instead receive its resources from activity 2. In doing so, activity 3 only starts one time period later than originally planned. Also, the heavily weighted activity 8 ($w_8 = 6$) now jumps in front of activity 6 ($w_6 = 1$), so that activity 8 can start as originally planned. Observe that all resource units are kept idle in time period 13. Continuing the

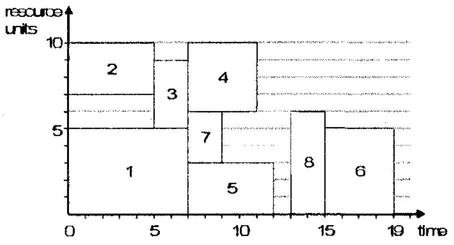


Figure 2.11. WET projected schedule at time instant 5

procedure for the disruption scenario of Table 2.1 yields the 19-period realized schedule of Figure 2.12 with a stability cost equal to 23.

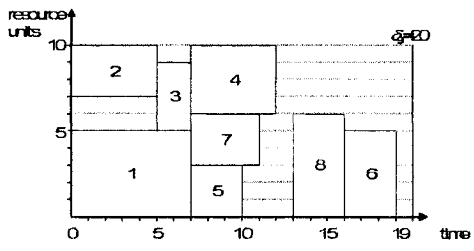


Figure 2.12. Realized schedule generated by the WET procedure

2.4.3 Complete rescheduling: The RCPSP procedure

The RCPSP procedure solves at each decision point the resource-constrained project scheduling problem associated with the projected remainder of the schedule. For the disruption scenario of our example problem, Figure 2.13 shows the realized schedule that was obtained by applying the branch-and-bound procedure originally used to derive the baseline schedule. It can again be observed that activity 8 is scheduled in front of activity 6, but this is not done in order to decrease the stability cost but to be able to finish the project as soon as possible. Activity 6 cannot start at time instant 11 due to the one-period disruption in activity 4, its predecessor in the network. Activity 5 finishes earlier than originally planned, creating the possibility for activity 8 to start at time instant 11. The project finishes at time 17 with a stability cost of 33.

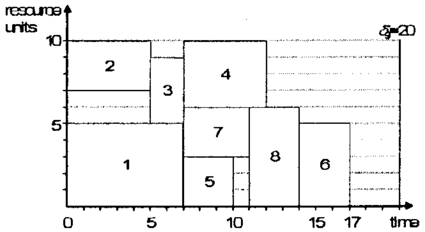


Figure 2.13. Realized schedule obtained by the RCPSP procedure

2.4.4 Activity-based priority rule

The *activity-based priority rule* (ABP) relies on an activity list deduced from the baseline schedule, i.e. the schedule shown in Figure 2.3. Ordering the activities in increasing order of their scheduled start time, the baseline schedule yields four possible activity lists (0,1,2,3,4,5,7,6,8,9), (0,1,2,3,4,7,5,6,8,9), (0,2,1,3,4,5,7,6,8,9) and (0,2,1,3,4,7,5,6,8,9). Using decreasing activity weights as tie-break rule yields the list (0,1,2,3,4,5,7,6,8,9). Applying a serial schedule generation scheme at each decision point generates the reactive schedule shown in Figure 2.14. The realized ABP schedule has a makespan of 18 time periods

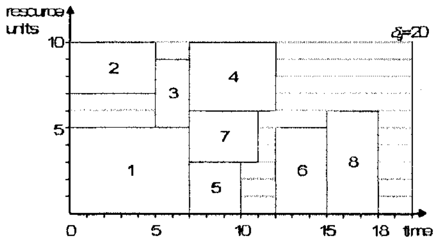


Figure 2.14. Realized schedule obtained by the ABP procedure

and a stability cost of 31. It differs from the schedule generated by the RCPSP procedure in that activity 6 which precedes activity 8 in the activity list, is scheduled as soon as possible at time instant 12 upon completion of activity 4. This prohibits activity 8 from starting before activity 6, although starting activity 8 before activity 6 results in a better makespan (see Figure 2.13) and lower stability cost (see Figure 2.12).

2.5 Experimental results on the makespan-stability trade-off

The above proactive-reactive procedures have been programmed in Microsoft Visual C++ 6.0 and tested on a set of eighty 30-activity networks generated by Van de Vonder et al (2005b) using the RanGen generator developed by Demeulemeester et al (2003). For an extensive description of the computational results we refer to Van de Vonder et al (2005bc, 2006). Here we limit ourselves to a summary of the main conclusions that can be drawn from these experiments.

Among the simple proactive baseline scheduling heuristics presented in this chapter, the STC heuristic ranks best on stability. Apparently, relying on the information provided by both the activity weights and the activity duration variability pays off. Improvement heuristics, such as tabu search, will typically yield better results but are subject to overfitting and will need substantially more computational effort.

Combining an exact procedure for generating a minimum makespan baseline schedule with an exact procedure for complete quality robust rescheduling when schedule breakage occurs excels in the best makespan performance in terms of the timely project completion probability $P(s_n \leq \delta_n)$, but is clearly outperformed in terms of stability cost measured by $\sum_j w_j E|s_j - s_j|$. Combining exact procedures for generating a minimum duration schedule with the reactive Fix Flow or ABP procedure is computationally far less demanding while yielding comparable makespan performance at smaller stability cost. Combining a minimum duration baseline schedule with a solution robust reactive procedure such as WET allows for acceptable makespan performance at still smaller stability cost, but requires again a high computational effort. Only when activity duration variability is very high and due dates are tight, the combination of minimum duration schedules and stability-improving reactive policies such as WET might result in unsatisfying makespan performance.

Activity duration variability has a strong impact on the obtained results. The lower the variability in the duration of the project activities, the more attractive solution robust baseline scheduling becomes. A project manager who anticipates only minor schedule disruptions and therefore decides to go for a deterministic quality robust baseline schedule, runs into a serious misconception of the impact of activity duration variability. Pure quality robust scheduling is inadequate when activity duration variability is low.

The very promising results obtained by the proactive baseline scheduling procedures that aim at generating solution robust (stable) baseline schedules, hold an invitation to continue the research on the development of stable baseline schedules. Especially when timely project completion is deemed important, when the activity duration variability is not too high and when the predefined project due dates are not too tight, leaving adequate room for buffer insertion, the use of proactive scheduling procedures that aim at generating solution robust (stable) schedules pays off.

Acknowledgments

This research has been supported by Project OT/03/14 of the Research Fund K.U.Leuven and project G.0109.04 of the Research Foundation-Flanders (FWO-Vlaanderen).

References

Artigues, C. and Roubellat, F. (2000). A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes, *European Journal of Operational Research*, 127:294-316.

Artigues, C., Roubellat, F. and Billaut J.C. (1999). Characterization of a set of schedules in a resource-constrained multi-project scheduling problem with multiple modes, *International Journal of Industrial Engineering*, 6:112-122.

Artigues, C., Billaut, J.C. and Esswein, C. (2005). Maximization of solution flexibility for robust shop scheduling, *European Journal of Operational Research*, 165:314-328.

Aytug, H., Lawley, M., McKay, K., Mohan, S. and Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161:86-110.

Billaut, J.-C., Moukrim, A. and Sanlaville, E. (Eds.) (2005). *Flexibilité et robustesse en ordonnancement*, Traité IC2, Série Informatique et systèmes d'information, Hermès-Lavoisier.

Billaut, J.C. and Roubellat, F. (1996a). A new method for workshop real-time scheduling, *International Journal of Production Research*, 34:1555-1579.

Billaut, J.C. and Roubellat, F. (1996b). Characterization of a set of schedules in a multiple resource context, *Journal of Decision Systems*, 5:95-109.

Blazewicz, J., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1983). Scheduling subject to resource constraints: Classification and complexity, *Discrete Applied Mathematics*, 5:11-24.

Davenport, A. and Beck, J. (2002). A survey of techniques for scheduling with uncertainty. Unpublished manuscript.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, *Management Science*, 38:1803-1818.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem, *Management Science*, 43:1485-1492.

Demeulemeester, E. and Herroelen, W. (2002). *Project scheduling - A research handbook*, International Series in Operations Research & Management Science, Vol. 49, Kluwer Academic Publishers, Boston.

Demeulemeester, E., Vanhoucke, M. and Herroelen, W. (2003). RanGen: A random generator for activity-on-the-node networks, *Journal of Scheduling*, 6:17-38.

Elmaghraby, S.E.E., Herroelen, W.S. and Leus, R. (2003). Note on the paper "Resource-constrained project scheduling using enhanced theory of constraints", *International Journal of Project Management*, 21:301-305.

Glover, F. (1989). Tabu search, Part I, *INFORMS Journal of Computing*, 1:190-206.

Glover, F. (1990). Tabu search, Part II, *INFORMS Journal of Computing*, 2:4-32.

Goldratt, E. (1997). *Critical chain*, The North River Press.

Hagstrom, J. (1988). Computational complexity of PERT problems, *Computers and Operations Research*, 18:139-147.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: notation, classification, models and methods" by Brucker et al, *European Journal of Operational Research*, 128:221-230.

Herroelen, W. and Leus, R. (2001). On the merits and pitfalls of critical chain scheduling, *Journal of Operations Management*, 19:559-577.

Herroelen, W. and Leus, R. (2004). The construction of stable baseline schedules, *European Journal of Operational Research*, 156:550-565.

Herroelen, W. and Leus, R. (2005). Project scheduling under uncertainty - Survey and research potentials, *European Journal of Operational Research*, 165:289-306.

Herroelen, W., Leus, R. and Demeulemeester E. (2002). Critical chain project scheduling - Do not oversimplify, *Project Management Journal*, 33:46-60.

Hoogeveen, H. (2005). Multicriteria scheduling, *European Journal of Operational Research*, 167:592-623.

Kolisch, R. (1996). Efficient priority rules for the resource-constrained project scheduling problem, *Journal of Operations Management*, 14:179-192.

Kouvelis, P. and Yu, G. (1997). *Robust discrete optimization and its applications*, Kluwer Academic Publishers, Boston.

Leus, R. (2003). *The generation of stable project plans - Complexity and exact algorithms*, Ph.D thesis, K.U.Leuven, Belgium.

Leus, R. and Herroelen, W. (2001). On the merits and pitfalls of critical chain scheduling, *Journal of Operations Management*, 19:559-577.

Leus, R. and Herroelen, W. (2004). Stability and resource allocation in project planning, *IIE Transactions*, 36:667-682.

Leus, R. and Herroelen, W. (2005). The complexity of machine scheduling for stability with a single disrupted job, *Operations Research Letters*, 33:151-156.

Mehta, S. and Uzsoy R. (1998). Predictive scheduling of a job shop subject to breakdowns, *IEEE Transactions on Robotics and Automation*, 14:365-378.

Möhring, R.H., Radermacher, F.J. and Weiss, G. (1984). Stochastic scheduling problems I - General strategies, *ZOR-Zeitschrift für Operations Research*, 28:193-260.

Möhring, R.H., Radermacher, F.J. and Weiss, G. (1985). Stochastic scheduling problems II - Set strategies, *ZOR-Zeitschrift für Operations Research*, 29:65-104.

Neumann, K., Schwindt, C. and Zimmermann, J. (2003). *Project scheduling with time windows and scarce resources: Temporal and resource-constrained project scheduling with regular and nonregular objective functions*, Springer-Verlag.

Newbold, R.C. (1998). *Project management in the fast lane: Applying the Theory of Constraints*, St. Lucie Press, New York.

Pinedo, M. (2002). *Scheduling. Theory, algorithms, and systems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Roy, B. (2002). Robustesse de quoi et vis-à-vis de quoi mais aussi robustesse pourquoi en aide à la décision?, *Proceedings of the 56th Meeting of the European Working Group on Multiple Criteria Decision Making* (Figueira, J., Henggeler-Anthunes, C. and Climaco, J. (Eds.)), Coimbra, Portugal.

Sanlaville, E. (2004). *Ordonnancement sous conditions changeantes*, Habilitation à Diriger des Recherches, Université Blaise Pascal, Clermont-Ferrand, France.

Stork, F. (2001). *Stochastic resource-constrained project scheduling*, Ph.D Thesis. TU Berlin, Berlin, Germany.

Tavares, L., Ferreira, J. and Coelho, J. (1998). On the optimal management of project risk, *European Journal of Operational Research*, 107:451-469.

Tukel, O.I., Rom, W.O. and Eksioglu, S.D. (2005). An investigation of buffer sizing techniques in critical chain scheduling. *European Journal of Operational Research*, to appear.

Van de Vonder, S., Demeulemeester, E., Leus, R. and Herroelen, W. (2005a). The use of buffers in project management: The trade-off between stability and makespan, *International Journal of Production Economics*, 97:227-240.

Van de Vonder, S., Demeulemeester, E. and Herroelen, W. (2005b). An investigation of efficient and effective predictive-reactive project scheduling procedures, *Journal of Scheduling*, to appear.

Van de Vonder, S., Demeulemeester, E. and Herroelen, W. (2005c). Heuristic procedures for generating stable project baseline schedules, Research Report 0516, Department of Applied Economics, K.U.Leuven, Belgium.

Van de Vonder S., Demeulemeester, E., Herroelen, W. and Leus, R. (2006). The trade-off between stability and makespan in resource-constrained project scheduling, *International Journal of Production Research*, 44:215–236.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W. (2001). An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem, *Annals of Operations Research*, 102:179-196.

Wu, S.D., Storer, H.S. and Chang, P.-C. (1993). One-machine rescheduling heuristics with efficiency and stability as criteria, *Computers and Operations Research*, 20:1-14.

Chapter 3

RESOURCE CONSTRAINED PROJECT SCHEDULING MODELS UNDER RANDOM DISTURBANCES

Dimitri Golenko-Ginzburg[1,3], Aharon Gonik[2], Anna Baron [3]
[1]*Academic College of Judea and Samaria, Ariel, 44837, Israel;*
[2]*Sapir Academic College, Doar Na Hof Ashkelon, 79165, Israel;*
[3]*Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel;*
dimitri@bgu.ac.il;aharong@mail.sapir.ac.il;malishev@bgu.ac.il

Abstract In the chapter we consider two different scheduling models for stochastic network projects. Models of the first type consider several simultaneously realized stochastic network projects of PERT type. Resource scheduling models of the second type also cover PERT type projects, but with two different kinds of renewable resources:

1 extremely costly resources (A-resources) which have to be obtained for a short time within the project's time span. Such resources have to be prepared and delivered externally at planned moments,
2 renewable resources (B-resources) which are at the company's disposal.

In all types of models each projects' activity utilizes several non-consumable related resources with fixed capacities, e.g. machines or manpower. For each operation, its duration is a random variable with given density function. The first problem centers on determining:

- the earliest starting moment for each project's realization,
- the limited resource levels for each type of resources to be stored during the projects' realization,
- the moments when resources are fed in and projects' activities start,

in order to minimize the average total expenses of hiring and maintaining resources subject to the chance constraints.

For the second class of developed models the problem boils down:

- to predetermine in advance, i.e., before each project starts to be realized, a deterministic delivery schedule for A-resources which are not at the project's disposal,
- to determine both the starting times and the resource capacities to be utilized for activities which require limited renewable B-resources which are at the project's disposal,
- to determine the starting moment of each project's realization,

in order to minimize average total projects' expenses subject to the chance constraint.

Problems of resource project scheduling are solved via simulation, in combination with a cyclic coordinate descent method and a knapsack resource reallocation model.

Keywords: Resource constrained project scheduling; Cyclic coordinate descent method; Resource reallocation; Resource delivery schedule; Non-consumable resources.

3.1 Introduction

In modern stochastic project management a company is usually faced with realizing several projects under random disturbances. The total amount of resources (usually the budget) at the company's disposal to carry out the projects is limited. The hierarchical control model at each level provides optimal decision-making as follows:

- at the company level decision-making boils down to optimal budget reassignment among the projects (Problem I, which is a planning procedure),
- at the project level optimal decision-making results either in optimal budget reallocation among the project's activities (Problem IIA serves as a control action) or in determining the optimal speed of the project's realization (Problem IIB which is a control action as well); thus, solving Problems IIA and IIB results in optimizing the progress of the project towards its goal, in order to reorient the project in the desired direction,
- at the inspection level, on-line control is carried out, i.e., optimal control points to inspect the progress of the project are determined (Problem III),
- at the lowest, the scheduling level, resource constrained project scheduling is realized by reallocating, if necessary, non-consumable resources among the project's activities (Problem IV, which is a scheduling procedure).

In the chapter we deal mostly with developing new models at the scheduling level. Several types of models are considered. Models of the first type consider several simultaneously realized stochastic network projects of PERT type. Resource scheduling models of the second type also cover PERT type projects, but with two different kinds of renewable resources:

1 extremely costly resources (A-resources) which have to be obtained for a short time within the project's time span. Such resources have to be prepared and delivered externally at planned moments,

2 renewable resources (B-resources) which are at the company's disposal.

In all types of models each projects' activity utilizes several non-consumable related resources with fixed capacities, e.g. machines or manpower. Each type of resource at the management's disposal is in limited supply, with a resource limit that is fixed at the same level throughout the projects' duration, i.e., until the last project is actually completed. For each operation, its duration is a random variable with given density function. Processing costs per time unit to hire and to utilize all the total available resources are pregiven. The problem is to determine:

- the earliest starting moment for each project's realization,
- the limited resource levels for each type of resources to be stored during the projects' realization,
- the moments when resources are fed in and projects' activities start,

in order to minimize the average total expenses of hiring and maintaining resources subject to the chance constraints.

For the second class of developed models the problem boils down:

1 to predetermine *in advance*, i.e., before each project starts to be realized, a deterministic delivery schedule for A-resources which are not at the project's disposal,

2 to determine both the starting times and the resource capacities to be utilized for activities which require limited renewable B-resources which are at the project's disposal,

3 to determine the starting moments of each project's realization,

in order to minimize average total projects' expenses subject to the chance constraint.

Problems of resource project scheduling are solved via simulation, in combination with a cyclic coordinate descent method and a knapsack resource reallocation model. The simulation model comprises three optimization cycles and can be used for small - and medium-size projects only. Otherwise aggregation has to be applied.

3.2 Notation

3.2.1 The company level (several controlled projects)

C	– total budget at the company's disposal to realize a group of projects,
$G_l(N, A)$	– the l-th stochastic network project of PERT or PERT-COST types, $1 \leq l \leq n$,
n	– number of projects,
$G_{lt}(N, A)$	– project $G_l(N, A)$ observed at moment $t \geq 0$,
D_l	– the due date of the l-th project (pregiven),
S_l	– the actual moment project $G_l(N, A)$ starts to be realized,
E_l	– the earliest moment for project $G_l(N, A)$ to start functioning (to be optimized and determined in advance),
F_l	– the actual moment project $G_l(N, A)$ is completed,
p_l	– desirable probability that in practice enables completion of the l-th project on time, i.e., $p_l \leq Pr\{F_t \leq D_l\}$ (pregiven).

3.2.2 Project level

$G(N, A)$	– activity-on-arc network project,
D	– the project's due date (pregiven),
p	– desirable probability that in practice enables completion of the project on time (chance constraint),
G_t	– project $G(N, A)$ observed at moment $t \geq 0$, $G_0 = G(N, A)$,
S	– time moment the project starts to be realized,
F	– time moment the project is accomplished,
S_{min}, S_{max}	– lower and upper bounds of the moment the project may actually start (pregiven).

3.2.3 Activity level

$(i, j) \subset G(N, A)$	– the project's activity,
t_{ij}	–random duration of activity (i, j) with density function $f_t(i, j)$,
S_{ij}	–the moment activity (i, j) actually starts,
$F_{ij} = S_{ij} + t_{ij}$	–the actual moment activity (i, j) is finished,
a_{ij}	–lower bound of t_{ij} (pregiven),
b_{ij}	–upper bound of t_{ij} (pregiven),
μ_{ij}	–average value of t_{ij} (pregiven),
$p(i, j)$	–conditional probability of activity (i, j) to be on the critical path in the course of the project's realization,

h	–number of activities entering the project,
$(i,j)_l$	–activity (i,j) entering the l-th project,
S_{ijl}	–the moment activity $(i,j)_l$ actually starts,
F_{ijl}	–the moment activity $(i,j)_l$ is finished,
t_{ijl}	–random duration of activity $(i,j)_l$,
a_{ijl}	–lower bound of value t_{ijl} (pregiven),
b_{ijl}	–upper bound of value t_{ijl} (pregiven),
μ_{ijl}	–average value of t_{ijl},
V_{ijl}	–variance of t_{ijl},
$p(i,j)_l$	–conditional probability of activity $(i,j)_l$ to be on the critical path in the course of the project's $G_l(N,A)$ realization (dependent on the decisions already taken),
$T(i)$	–the moment event (node) i is realized, i.e., the earliest moment when all activities entering i are completed (a random value),
$D(i)$	–the subset of nodes which *directly precedes* node i, i.e., $i^* \in D(i)$ means that activity (i^*,i) enters $G(N,A)$.

3.2.4 Resource parameters

r_{ijq}	– capacity of the q-th type non-consumable resource(s) allocated to activity $(i,j), 1 \leq q \leq m$,
m	– number of different resources,
R_q	– total available non-consumable resources of type q at the disposal of the project management throughout the planning horizon, satisfying $R_{q\,min} \leq R_q \leq R_{q\,max}$,
$R_{q\,min}, R_{q\,max}$	– lower and upper bounds of the resource level R_q, $1 \leq q \leq m$, (determined by the project management and externally pregiven),
c_q	– cost per time unit to hire and to utilize the q-th resource unit,
E_0	– the moment for resources $\{R_q\}$ to be hired and delivered, i.e., the moment the system starts functioning,
$R_q(t)$	– free available non-consumable resources of type q at moment $t \geq 0$,
$R_q(t \mid S_{ijl})$	– maximal value of the q-th resource profile at moment t on condition that activities $(i,j)_l$ start at moment S_{ijl},
$T(G_l \mid S_{ijl})$	– random duration of project $G_l(N,A)$ on the same condition,
(i_{ξ_A}, j_{ξ_A})	– activity which utilizes A-resources, $1 \leq \xi \leq h_A < h$
h_A	– number of activities which have to be supplied with

	A-resources,
$(i_{\omega_B}, j_{\omega_B})$	– activity which utilizes B-resources, $1 \leq \omega \leq h_B < h$,
h_B	– number of activities which have to be supplied with B-resources,
$r_{i_{\omega_B}, j_{\omega_B}, q}$	– capacity of the q-type B-resources to be allocated to activity $(i_{\omega_B}, j_{\omega_B})$,
r_{ijlq}	– capacity of the q-type resources allocated to activity (i,j) entering project $G_l(N,A)$,
$T(i_{\xi_A}, j_{\xi_A})$	– time moments A-resources have to be delivered to process activity (i_{ξ_A}, j_{ξ_A}) (an optimal deterministic schedule to be determined in advance),
$S(i_{\xi_A}, j_{\xi_A})$	– time moments activity (i_{ξ_A}, j_{ξ_A}) actually starts,
$c(i_{\xi_A}, j_{\xi_A})$	– penalty cost per time unit for the idleness of A-resources (in case $S(i_{\xi_A}, j_{\xi_A}) > T(i_{\xi_A}, j_{\xi_A})$).

3.3 Scheduling several stochastic network projects of PERT type

3.3.1 The system's description

It can be well-recognized that there is a lack of literature on resource supportability models (see Bell and Han (1991), Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998), Golenko-Ginzburg et al (2000), Golenko-Ginzburg et al (2000a), Golenko-Ginzburg et al (2001), Gonik (1995), Gonik (1999), Kolisch (1995), Luss (1991), Sitniakovski (2002), Toker et al (1991), Voropajev et al (1999), Zhan (1994). Besides (see Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998), Golenko-Ginzburg et al (2000), Golenko-Ginzburg et al (2000a), Golenko-Ginzburg et al (2001)), no published resource constrained project scheduling algorithm considers stochastic network projects, e.g. PERT-type projects. Since, in practice, PERT projects are usually carried out with limited resources, the need for proper resource supportability models for PERT network projects is very high.

In papers Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998), Golenko-Ginzburg et al (2000), Golenko-Ginzburg et al (2000a), Golenko-Ginzburg et al (2001) a network project of PERT type with random activity durations and several non-consumable limited resources is considered. For each type of resource q, its limit remains unchanged throughout the project's duration. A project's activity requires resources of various types with either constant (see Golenko-Ginzburg and Gonik (1997)) or variable capacities (see Golenko-Ginzburg and Gonik (1998)) and is operated at a random speed. The problem is to determine for each activity (i,j) its starting time S_{ij}, i.e., the timing of feeding-in-resources for that activity. The objective is to minimize the expected project's duration.

It can be clearly recognized that both resource supportability models (see Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998)) fit only certain project management scenarios. Those models do not include cost objectives, i.e., the costs of hiring and maintaining resources throughout the project's realization. The models do not deal with projects' due dates as well as with chance constraints of meeting the projects' deadlines on time. Those models can be used for one project only.

The chapter under consideration considers a more generalized resource supportability model in project management (see Golenko-Ginzburg et al (2000), Golenko-Ginzburg et al (2000a), Golenko-Ginzburg et al (2001), Gonik (1995), Gonik (1999), Sitniakovski (2002)).

Several simultaneously realized stochastic network projects $G_l(N, A)$ of PERT type are considered. The durations of all projects' activities are random and the corresponding probability density functions are pregiven. Each activity requires various types of renewable resources with fixed capacity. Resources are stored and maintained at one central warehouse; each type of resources is in limited supply and is fixed at the same level throughout the projects' realization. Resources are to be hired and delivered to the central store before the moment the first project starts to be realized. They are released at the moment when the last project is completed. Each activity starts at the moment when it is ready to be processed and when free available resources can support it. The cost of hiring and monitoring a resource unit per time unit (for each type of resources) is pregiven. Each project has its due date and the least permissible probability of accomplishing the project on time, i.e., its chance constraint. The problem is to determine:

- the earliest starting moment E_l for each project's realization, $1 \leq l \leq n$;
- the limited resource levels R_q, $1 \leq q \leq m$, for each type of resources to be stored during the projects' realization,
- the moments S_{ijl}, that resources are fed in and projects' activities start,

in order to minimize the average total expenses of hiring and maintaining resources subject to the chance constraints.

Thus, the developed resource supportability model covers a flexible project management system. The model minimizes the average operational expenses subject to the chance constraints, for each project separately. The problem is solved via simulation. Two optimization cycles are imbedded in the model. The external cycle deals with optimizing both the projects' earliest starting moments together with the resource levels. Those parameters serve as the input values for the internal cycle. The latter uses heuristic decision-making rules to reallocate free available resources among the projects in order to meet the projects' chance constraints. Note that models (see Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998)) are based on knapsack resource reallocation problems which are applied at decision points when at least one activity is ready to be operated and there are free available resources. If, at a certain point of time, a set of more than one activity is ready to be operated but the available amount of resources is insufficient, a competition among the activities is carried out in order to choose a subset of those activities which has to be operated first and can be supplied by the available resources. Determining such an optimal subset of activities is carried out via a knapsack problem. However, for *several* stochastic network projects the corresponding knapsack problem becomes too complicated. We have substituted it by a heuristic decision-making procedure. Note that the developed resource supportability model is a very complicated stochastic optimization problem which cannot be solved in the general case and allows only a heuristic solution.

3.3.2 The problem

The problem is to determine values $E_l, 1 \leq l \leq n, R_q, 1 \leq q \leq m$, and S_{ijl} to minimize the expected total resource expenses (see Notations 3.2.1, 3.2.2, 3.2.3 and 3.2.4)

$$\bar{C} = \min_{E_l, R_q, S_{ijl}} E\left\{ \left[\sum_{q=1}^{m} (R_q \cdot c_q) \right] (\max_l F_l - \min_l S_l) \right\} \tag{3.1}$$

subject to

$$P\{F_l \leq D_l\} \geq p_l, 1 \leq l \leq n, \tag{3.2}$$

$$R_q(t|S_{ijl}) \leq R_q(t) \leq R_q, \forall t \geq 0, 1 \leq q \leq m. \tag{3.3}$$

Model (16.1-16.3) is a very complicated stochastic optimization problem which cannot be solved in the general case; the problem only allows a heuristic solution.

The most widely used PERT techniques (see, e.g. Golenko (1972)) are based on assumptions that each activity duration t_{ijl} follows either a normal probability density distribution with parameters (μ_{ijl}, V_{ijl}) or an uniform distribution in the interval (a_{ijl}, b_{ijl}), or a beta probability density function

$$f_{ijl}(x) = \frac{12}{(b_{ijl} - a_{ijl})^4}(x - a_{ijl})(b_{ijl} - x)^2. \tag{3.4}$$

Note that obvious relations

$$\max_{i,j,l,q} r_{ijlq} \leq R_q, 1 \leq q \leq m, \tag{3.5}$$

hold, otherwise the projects cannot be operated.

The basic idea of the heuristic solution is as follows. Two levels are incorporated in the model - the upper (external) level and the lower level. At the upper level an approximate search algorithm is used to determine the optimal values E_l and $R_q, 1 \leq q \leq m, 1 \leq l \leq n$. We suggest to use the cyclic coordinate descent method (see Luenberger (1973)) which is simple in usage and has been realized in various production control and project management problems (see Gonik (1999), Sitniakovski (2002)). Parameters $\{E_l, R_q\}$ serve as the input values for the lower level where values S_{ijl} are determined via simulation. Decision-making is carried out at essential moments t, either

- when one of the activities $(i,j)_l$ is finished at moment F_{ijl} and additional amount of resources $r_{ijlq}, 1 \leq q \leq m$, becomes available, or
- when all activities (i,j) belonging to one and the same project $G_l(N, A)$ and leaving node i are ready to be processed, or
- when several subsets of activities ready to be processed belong to different projects.

If one or more activities are ready to be processed at moment t and *all of them* can be supplied with available resources, the needed resources are fed in and the activities start to be operated at moment t, i.e. $S_{ijl} = t$. If at least for one type q of resources there is a lack of available resources at moment t, a competition among the activities has to be arranged to choose a subset of activities that will start to be operated at moment t and can be supplied by resources. The general idea of decision-making, i.e., the sub-problem of choosing activities to be operated, will be outlined below.

3.3.3 Heuristic decision-making

Assume that at a certain moment t a set of $\gamma > 1$ activities $(i_1, j_1)_{l_1}, \ldots, (i_\gamma, j_\gamma)_{l_\gamma}$ is ready to be operated. Two cases will be considered:

1 All γ activities under consideration enter one and the same network graph with one and the same index, i.e., equality $l_1 = \ldots = l_\gamma$ holds.

2 Activities $(i_1, j_1)_{l_1}, \ldots, (i_\gamma, j_\gamma)_{l_\gamma}$ refer to more than one network graph, i.e., for at least one couple of activities $[(i_v, j_v)_{l_v}, (i_w, j_w)_{l_w}], 1 \leq v, w \leq \gamma$, inequality $v \neq w$ holds.

Let us examine both cases in greater detail.

Case A. To simplify the problem, cancel parameter l since the latter remains unchanged in the course of decision-making. Assume (see Golenko-Ginzburg and Gonik (1997)), that at moment t, γ activities $(i_1, j_1), \ldots, (i_\gamma, j_\gamma), \gamma > 1$, are ready to be processed, and at least for one type q of resources there is a lack of available resources, i.e., relation

$$\sum_{\xi=1}^{\gamma} r_{i_\xi j_\xi q} > R_q(t) \tag{3.6}$$

holds. Here R_{ijq} is a simplified modification of r_{ijql} for a fixed l. A competition among the activities is arranged following the heuristic (see Golenko-Ginzburg and Gonik (1997)). According to that heuristic, the subset, which provides the maximal total contribution to the expected project duration subject to 16.6, has to be chosen. Each activity (i, j) contributes to the expected project duration value $\vartheta_{ij} = \mu_{ij} \cdot p(i, j)$, where $p(i, j)$, being a simplified version of $p(i, j)_l$, is the conditional probability for the activity (i, j) to be on the critical path. At any decision point t values $p(i, j)$ are calculated via simulation (see Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998)). After obtaining values $p(i_\xi, j_\xi)$, $1 \le \xi \le \gamma$, for all competitive activities at moment t, the optimal subset is chosen by solving a zero-one integer programming problem as follows: determine integer values η_{i_ξ, j_ξ}, $1 \le \xi \le \gamma$, to maximize the objective

$$\max_{\{\eta_{i_\xi j_\xi}\}} \left\{ \sum_{\xi=1}^{\gamma} \left[\eta_{i_\xi, j_\xi} \cdot \mu_{i_\xi, j_\xi} \cdot p(i_\xi, j_\xi) \right] \right\} \tag{3.7}$$

subject to

$$\sum_{\xi=1}^{\gamma} \left(\eta_{i_\xi j_\xi} \cdot r_{i_\xi j_\xi q} \right) \le R_q(t), 1 \le q \le m, \tag{3.8}$$

where

$$\eta_{i_\xi j_\xi} = \begin{cases} 0 & \text{if activity } (i_\xi, j_\xi) \text{ will not obtain resources} \\ 1 & \text{otherwise} \end{cases} \tag{3.9}$$

Problem 16.7-16.9 is a classical zero-one integer programming problem, which provides a precise solution. However, the problem's parameters, e.g. ϑ_{i_ξ, j_ξ}, are obtained via heuristic assumptions.

Case B. This case makes unable decision-making 16.7-16.9 since the latter does not take into account at moment t different projects $G_{lt}(N, A)$ with different due dates D_l and different chance constraints p_l. Assume that at moment t a set of activities which are ready to be processed and which belong to φ different projects $G_{l_\theta t}(N, A)$, $1 \le \theta \le \varphi$, is given. This set of activities can be subdivided into φ subsets $\{(i_{\xi_\theta}, j_{\xi_\theta})_{l_\theta}\}_\theta$, $1 \le \xi \le \gamma$ each subset of volume γ_θ entering the project $G_{l_\theta t}(N, A)$. Assume, that there is a lack of available resources, i.e., at least for one type q of resources relation

$$\sum_{\theta=1}^{\varphi} \sum_{\xi=1}^{\gamma_\theta} r_{i_{\xi_\theta} j_{\xi_\theta} l_\theta q} > R_q(t) \tag{3.10}$$

holds.

In order to undertake a reasonable decision-making, i.e., to choose a quasi-optimal subset of activities, we suggest a heuristic step-by-step procedure. The procedure is realized as follows:

Step 1. For each project $G_{l_\theta t}(N, A)$ separately, reorder the activities entering the subset $\{(i_{\xi_\theta}, j_{\xi_\theta})_{l_\theta}\}_\theta$ in the descending order of their corresponding values $\vartheta_{i_{\xi_\theta} j_{\xi_\theta}} = \mu_{i_{\xi_\theta} j_{\xi_\theta}} \cdot p(i_{\xi_\theta}, j_{\xi_\theta})_{l_\theta}$.

Step 2. An assumption is introduced that:

- project $G_{l_1 t}$ will not obtain at moment t the needed resources for any of the q_l activities $\{(i_{\xi_1}, j_{\xi_1})_{l_1}\}$ ready to be processed;
- the needed resources will be fed in for all activities $\{(i_{\xi_1}, j_{\xi_1})_{l_1}\}$ at the next decision moment t^*. Value t^* can be calculated as the *minimal* value of the average finishing times of all activities which at moment t undergo processing;
- in future, i.e., at all decision points $t' > t$, all the remaining activities $(i, j)_{l_1}$ belonging to that projects will not wait for resources in lines until the end of the project's realization.

Via simulation calculate the project's random duration $T(G_{l_1} | S_{ijl})$ honoring the outlined above assumptions.

Step 3. Realize step 2 M times in order to obtain a representative statistics. Call the random finishing times for project $G_{l_1}(N, A)$: $F_{l_1}^{(1)}, F_{l_1}^{(2)}, \dots, F_{l_1}^{(M)}$.

Step 4. Calculate the statistical frequency Q_{l_1} of completing project $G_{l_1}(N, A)$ on time:

$$Q_{l_1} = \frac{\sum_{\alpha=1}^{M} W_{l_1}^{(\alpha)}}{M} \tag{3.11}$$

where

$$W_{l_1}^{(\alpha)} = \begin{cases} 1 & \text{if } F_{l_1}^{(\alpha)} \le D_{l_1} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

Step 5. Calculate the relative deviation

$$Z_{l_1} = \left(Q_{l_1} - p_{l_1}\right) \cdot \frac{1}{p_{l_1}} \tag{3.13}$$

Step 6. Repeat steps 2$\longrightarrow$5 for all projects $G_{l_\theta}(N, A), 1 \le \theta \le \varphi$, under competition.

Step 7. Choose the project with the *lowest* value Z_θ. Let it be $G_{l_\omega}(N, A), 1 \leq \omega \leq \varphi$.

Step 8. For project $G_{l_\omega}(N, A)$, all the sorted activities $(i_{\xi_\omega}, j_{\xi_\omega})_{l_\omega}$ (see step 1) are examined one after another, in the descending order of their priorities, from top to bottom, to determine *the first* activity, which can be supplied with available resources. If, for such an activity $(i_{\xi_\omega}, j_{\xi_\omega})_{l_\omega}, 1 \leq \xi \leq \gamma_\omega$, relations $r_{i_{\xi_\omega}, j_{\xi_\omega}, l_\omega, q} \leq R_q(t), 1 \leq q \leq m$, hold, the needed resources are passed to the activity while the available resources $R_q(t)$ are updated, $R_q(t) - r_{i_{\xi_\omega}, j_{\xi_\omega}, l_\omega, q} \Rightarrow R_q(t), 1 \leq q \leq m$.

If such an activity can be obtained, go to step 10. Otherwise apply the next step.

Step 9. If no activity $(i_{\xi_\omega}, j_{\xi_\omega})_{l_\omega}$ can be chosen in the course of realizing step 8, take the next project with the lowest value Z_θ (besides Z_ω) in order to examine that project as well, etc., until a certain activity $(i_{\xi_\theta}, j_{\xi_\theta})_{l_\theta}$ will be determined. If no activity can be found by examining all the projects, go to step 11. Otherwise apply the next step.

Step 10. Exclude the determined activity from the set of competitive activities; update the available resources. Go to step 1, i.e., start realizing decision-making anew.

It can be well-recognized that the procedure terminates either when all the available resources are reallocated among activities or all the competitive projects are examined in the order of their emergency parameters Z_θ.

Step 11. Calculate the next decision point $t' > t$. Determine the set of activities ready to be operated. Go to step 1.

A conclusion can be drawn that in case B decision-making centers on choosing and operating first the activities which enter the "weakest" projects, i.e., the projects being late with meeting their corresponding due dates on time subject to their chance constraints. As to case A, the project management operates first the optimal subset of activities that provides minimization to the expected project's duration.

3.3.4 The structure of the resource supportability model

The initial data of the model is as follows:

at the company level: resource cost parameters $c_q, i \leq q \leq m$;

at the project level: due dates D_l and chance constraints $p_l, 1 \leq l \leq n$;

at the activity level: upper and lower bounds b_{ijl} and a_{ijl}, average values μ_{ijl}, resource capacities r_{ijlq}.

Decision variables $R_q, 1 \leq q \leq m$, and $E_l, 1 \leq l \leq n$, have to be determined *beforehand*, i.e., before the projects will actually start to be realized. Note that moment E_0 resources R_q have to be hired, delivered and stored at the company's central warehouse satisfies $E_0 = \min_l E_l$ and coincides with the beginning of the projects' realization. However, certain projects may start to be realized later that at moment E_0.

Thus, the resource supportability model is realized at two stages:

- at the *planning stage*, i.e., before the projects' realization, when determining optimal planning parameters E_l and $R_q, 1 \leq l \leq n, 1 \leq q \leq m$. Those parameters are input values for the *stage of monitoring* which is performed in the course of projects' realization.
- at the stage of *monitoring* the resource feeding-in-moments S_{ijl} are determined. Those parameters cannot be predetermined since they are random values conditioned on our future decisions. At the stage of monitoring resource supportability model can be realized in real time; namely, all activities can be operated only after obtaining necessary resources. However, if we want to evaluate the efficiency of the resource supportability model, we can simulate the algorithm's work by random sampling of the actual duration of activities. By simulating the algorithm's work many times, all the projects' cost and probability parameters can be evaluated.

The structure of the resource supportability model and its algorithm is based on the assertion, that the cost objective $\bar{C}$ is a complicated non-linear function of decision variables E_l and $R_q, 1 \leq l \leq n, 1 \leq q \leq m$, and, by introducing the outlined above decision-making rules for cases A and B, is *fully determined* by those decision variables. Thus, it is reasonable to arrange two optimization cycles for the model:

- the external cycle to realize an optimal search for values $\{E_l\}$ and $\{R_q\}$ by using the cyclic coordinate descent method and
- the internal cycle to realize mutual simulation runs of the projects' realization with input values $\{E_l\}$ and $\{R_q\}$ obtained from the external cycles. It goes without saying that decision-making rules for both cases A and B are incorporated in the simulation model at the internal cycle. At each simulation run objective C is calculated.

The combination $\{R_q, E_l\}$ which provides the minimal average objective $\bar{C}$ calculated by (16.1), subject to all chance constraints (16.2) is taken as an optimal combination which has to be predetermined before the projects' realization. Needed resources $\{R_q\}$ are hired at the moment $E_0 = \min_l S_l$, after which the projects' realization actually starts. Feeding-in resource moments S_{ijl} are determined either for real-time projects, or by simulating the projects' realization.

3.3.5 Heuristic algorithm

The enlarged step-by-step procedure of the algorithm is as follows:

Step 1. Set the initial (minimal) values of $\{E_l\}$ and $\{R_q\}$. Note that $\{R_q\}$ are restricted from below:

$$R_q \geq \max_{i,j,l} r_{ijlq}, 1 \leq q \leq m, \tag{3.14}$$

otherwise the problem has no solution. For most practical cases values $E_l, 1 \leq l \leq n$, can be set equal zero. Thus, the optimal search method has to be arranged in the $(m+n)$ - dimensional area. Denote the initial $(m+n)$- dimensional search point by $X^{(0)}$.

Step 2. Realize a cyclic coordinate search method with a positive search step increment Δt (or ΔR_q), beginning from the initial search point $X^{(0)}$. Undertaking a search means shifting one of the coordinates, beginning from E_l (the first group of n coordinates $\{E_l\}$ has to precede the second group $\{R_q\}$) to the right with step Δt or ΔR. If, e.g., from the search point $X^{(\tau)}$ the search $X^{(\tau)} \longrightarrow X^{(\tau+1)}$ results in changing the λ-th coordinate, $1 \leq \lambda \leq m+n$, then all other coordinates remain unchanged. If in the course of a search step objective $\bar{C}$ becomes less than it has been before, at point $X^{(\tau)}$, the search proceeds in the same direction, i.e., an additional increment Δt (or ΔR_q) is realized. If the objective does not decrease, then we examine the next, $(\lambda+1)$-th coordinate, while all λ preceding coordinates remain unchanged with the values they have already received. The routine iteration of the search terminates when all $(m+n)$ coordinates $\{E_l\}$ and $\{R_q\}$are examined. Thus, each decision variable is *optimized separately*, while all the previous coordinates have already been optimized.

Step 3. At each routine search point $X^{(\tau)}$ with decision variables $\{S_l^{(\tau)}, R_q^{(\tau)}\}$, numerous simulation runs using the simulation model at the internal cycle have to be undertaken to obtain representative statistics for value $\bar{C}$. The simulation model comprises three submodels as follow:

Submodel I simulates most of the procedures to be undertaken in the course of the projects' realization, namely:

- determines decision points (essential moments);
- singles out (at a routine decision point) all activities that are ready to be operated;
- if possible, supplies all those activities with available resources and later on simulates the corresponding activities' durations;
- returns the utilized non-consumable resources to the company's central warehouse (at the moment an activity is finished);
- updates the remaining projects (if necessary) at each routine decision point.

Submodel II calculates via simulation values $p(i,j)$ to realize decision-making for the case of one project (case A), as well as values $p(i,j)$ for the case of several projects (case B). Submodel II calculates as well the forecasted value t^* of the next adjacent decision point (see step 2 of the decision-making model outlined in Section 3.3). For each activity (i,j) which at moment t is realized but has not been completed as yet, the average finishing time $\bar{F}_{ijl}$ is calculated. Given the starting time S_{ijl}, the probability density function of random value t_{ijl} and decision point t under consideration, a precise determination of value $\bar{F}_{ijl}$ can be obtained.

Note that simulation of activity durations by using Submodel II is carried out to solve auxiliary forecasting problems, but not to simulate actual activity realizations. The latter is carried out only by Submodel I.

Submodel III solves, at a routine decision point t, the zero-one integer programming problem (16.7-16.9) to undertake decision-making in the case of one project. Submodel III also simulates steps 3-9 of the decision-making model in case B of several projects (see 4.3.3).

The outcome value of the simulation model at step 3 is calculated as follows:

$$C = \frac{1}{M}\sum_{\delta=1}^{M}\left\{\sum_{q=1}^{m}(R_q \cdot c_q)[\max_l F_l^{(\delta)} - E_0]\right\} + \sum_{l=1}^{n}(A \cdot X_l). \qquad (3.15)$$

Here A is an essentially high value (for numerical examples we usually set A equal 10^{17}), while X_l satisfies

$$X_l = \begin{cases} 1 & \text{if } Q_l < p_l \\ 0 & \text{otherwise,} \end{cases} \qquad (3.16)$$

where Q_l is calculated by (3.11) and F_l^{δ} is the simulated moment project $G_l(N,A)$ is finished in the δ-th simulation run, $1 \le \delta \le M$.

Thus, relations (3.15-3.16) enable undertaking search for routine $(m+n)$-dimensional points X^{τ} honoring chance constraints (16.2). If at least one value $X_l = 1$, the corresponding combination $X^{(\tau)\equiv\{E_l,R_q\}}$ is withdrawn from the cyclic coordinate search process.

Step 4. After optimizing all $(m+n)$ coordinates $\{E_l\}$ and $\{R_q\}$, i.e., undertaking a routine search iteration, the search process is carried out anew, beginning from the first coordinate E_l. The search process terminates when, for two adjacent iterations f and $f+1$, the relative difference between $\bar{C}^{(f)}$ and $\bar{C}^{(f+1)}$ is less than the pregiven accuracy $\epsilon > 0$.

Extensive experimentation for medium size network projects have illustrated the efficiency of the developed two-level algorithm. Two iterations are usually enough to realize the optimization process.

3.4 Resource delivery schedules for PERT type network projects

3.4.1 The system's description

A number of recent papers have demonstrated the advantage of developing the planning and control models on resource constrained project scheduling (see 3.3.1), including a variety of publications on stochastic network projects (see Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998), Golenko-Ginzburg et al (2000), Golenko-Ginzburg et al (2000a), Golenko-Ginzburg et al (2001), Gonik (1995), Gonik (1999), Sitniakovski (2002)). Since, in practice, many projects, especially in R & D, are carried out under random disturbances, developing new models on resource constrained project scheduling with indeterminacy, remains an important problem in project management (PM), both from the theoretical and applied points of view.

A stochastic network project of PERT type is considered. The duration of each activity is random and the corresponding probability density function is pregiven. Certain activities entering the project require extremely costly and rare resources (A-resources) which are usually delivered externally and are available for short periods within the time span of the project (e.g. technical experts, test-benches, special and unique facilities, heavy duty equipment and cranes, etc.). A-resources should be strictly monitored because shortages might significantly affect the project schedule. Although it is unknown in advance when a certain activity which utilizes A-resources, will actually be ready to start, A-resources have to be *delivered at a pregiven date* that has to be determined in advance. Thus, for activities, which utilize A-resources, a *deterministic* schedule of delivering resources is to be predetermined before the project starts to be realized.

Other activities require constrained renewable resources (B-resources) which are at the disposal of the project management and are in limited supply for each type of resources. Assume that a resource limit is independent on time, i.e., is fixed at the same level throughout the project's duration. Various B-resources, e.g. skilled workers, special equipment, etc., for projects under random disturbances require flexible, but not close, monitoring. Since each activity entering the project is of random duration, the corresponding feeding-in resource moment to be determined is a random value too.

Note, that an activity may utilize several non-consumable (renewable) B-resources of various types with fixed (pregiven) capacities.

B-resources have to be hired in advance, in order to be delivered to the project management's store at the moment the project actually starts. B-resources are released at the moment when the project is completed. The B-resource limits for each type of resources are problem's variables to be optimized as well as the moment the project starts to be realized.

The cost objective of the control model comprises the following expenses:

1 The costs of hiring and maintaining B-resources within the project's duration i.e., between the moment the project starts to be realized and the moment of the project's completion.

2 The cost penalties paid for the A-resource idleness when an A-resource was delivered at the planned moment but not utilized since it had to wait for the moment the corresponding activity was ready to be operated.

3 The project has it's due date D and the penalty cost C^* (paid to the customer) for not accomplishing the project on time. In addition a penalty cost C^{**} has to be charged for each time unit of delay after the due date. If the project is accomplished before D, it has to be stored until the due date with a C^{***} penalty charge for each time unit of storage.

Note that the *operational costs* of processing project's activities are not implemented in the cost objective. This is done deliberately since all operational expenses remain unchanged and do not depend on the control model.

Besides papers Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998), Golenko-Ginzburg et al (2000), Golenko-Ginzburg et al (2000a), Golenko-Ginzburg et al (2001), Gonik (1999), all publications on managing resources for stochastic network projects are not generalized and fit only specific project scenarios. However, even models developed by Golenko-Ginzburg and Gonik (1998), and Gonik (1999) comprise several drawbacks as follows:

1 The generalized resources constrained scheduling model (see Gonik (1999)) deals with two different types of renewable resources which are consumed by the project's activities:

- rare and costly resources (call it henceforth A-resources) which have to be delivered from outside for a relatively small group of project activities;
- restricted renewable resources which are feed in at random moments when the resources are available and at least one project activity has to be supported with resources in order to start processing (B-resources). Those resources are in limited supply at the project's disposal throughout the planning horizon.

The model (see Gonik (1999)) honors the assumption that the total B-resource capacities for the project management store are fixed and pre-given externally. However, since the cost of hiring and maintaining B-resources is an essential part of the total expenses in the course of the project's realization, the problem of determining the optimal restricted B-resource capacity limits is reasonable for many projects' scenarios.

2 Minimizing the total project's expenses to meet the target on time, i.e., at a given due date, has not to be the only PM's goal in the course of a long-term cooperation with various customers. To honor the company's good name, an additional requirement has to be implemented in the model: the project has to meet its due date on time with a pregiven confidence probability. Thus, a chance constraint has to be introduced in the resource constrained model.

3 The cost objective for both outlined above models is to minimize the budget for the resource consumption within the planning horizon. However, it would be more reasonable to take into account additional factors connected with the project's total expenses within the planning horizon, e.g.:

   - the starting time of the project's realization, which refers to the optimized variables as well,
   - various penalty costs for not meeting the project's target on time and storage costs for the project's completion before the due date.

Thus, developing a generalized resource supportability model under a chance constraint and comprising al the additional parameters outlined above results in raising the model's flexibility. Such a model covers a broad spectrum of PM systems.

Let us formulate the essence of the developed resource supportability models (see Golenko-Ginzburg et al (2000), Golenko-Ginzburg et al (2000a), Golenko-Ginzburg et al (2001)) which is a further extension of recent publications (see Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998), Gonik (1999), Sitniakovski (2002)).

Given:

- the project's due date D;
- the least permissible probability p of accomplishing the project on time;
- the cost per time unit for hiring and maintaining a B-resource unit (for each type of resources);
- the penalty cost $c(i, j)$ per time unit for the idleness of A-resources (for each activity (i, j) which utilizes those resources) (see Notation);
- the penalty cost for the project's delay (a single payment to the customer);
- the penalty cost for each time unit of delay;
- storage charges per time unit for the project's completion before the due date,

the problem is to determine :

- the starting moment S of the project's realization, together with
- the resource delivery schedule for A-resources, and
- the restricted resource levels for each type of B-resources, in order to minimize the average total project's expenses subject to the chance constraint.

The problem is solved via simulation, in combination with a cyclic coordinate descent method and a knapsack resource reallocation model. The simulation model comprises two optimization cycles.

3.4.2 The problem

The general problem is as follows:
to determine *in advance* optimized deterministic variables $S, R_q, 1 \le q \le m$, and $T(i_{\xi_A}, j_{\xi_A}), 1 \le \xi \le h_A$, and, *within the project's realization*, actual starting times S_{ij} for all activities $(i,j) \in G$ (random values), in order to minimize the average project's expenses

$$\min_{\{S,R_q\},\{T(i_{\xi_A},j_{\xi_A})\}} \bar{C} \tag{3.17}$$

subject to

$$Pr\{F \le D\} \ge p, \tag{3.18}$$

$$S_{i_{\xi_A} j_{\xi_A}} \ge T(i_{\xi_A}, j_{\xi_A}), 1 \le \xi \le h_A, \tag{3.19}$$

$$S_{ij} \ge T(i) \, \forall (i,j) \in G, \tag{3.20}$$

$$t = S_{i_{\omega B} j_{\omega B}}, 1 \le \omega \le \gamma \le h_B, \Rightarrow \sum_{\omega=1}^{\gamma} r_{i_{\omega B} j_{\omega B} q} \le R_q(t), 1 \le q \le m, \tag{3.21}$$

where random value C satisfies

$$\begin{aligned} C \;=\; & \sum_{(i_{\xi_A}, j_{\xi_A})} \left\{ c(i_{\xi_A}, j_{\xi_A}) [S_{i_{\xi_A} j_{\xi_A}} - T(i_{\xi_A}, j_{\xi_A})] \right\} \\ & + \sum_{q=1}^{m} [c_q \cdot R_q \cdot (F - S)] \\ & + [C^* + C^{**}(F - D)]\delta + C^{***}(D - F)(1 - \delta) \end{aligned} \tag{3.22}$$

and

$$\delta = \begin{cases} 1 & \text{if } F > D \\ 0 & \text{otherwise.} \end{cases} \tag{3.23}$$

Restriction (3.19) means that an activity which utilizes A-resources, cannot start before its corresponding delivery moment. Restriction (3.20) means that any activity (i, j) entering G, cannot start before the moment $T(i)$, i.e., that (see Notation)

$$S_{ij} \geq \max_{i^* \in D(i)} \{S_{i^*i} + t_{i^*i}\} \tag{3.24}$$

holds. Restriction (15.1) means that if at a certain decision-point t B-resources are reallocated among $\gamma < h_B$ activities, the summarized value of supplied resources (for each q-type of B-resources) must not exceed the corresponding value $R_q(t)$, i.e., the total capacity of free available q-type resources at moment t, $1 \leq q \leq m$.

Problem (3.17-3.23) is a complicated stochastic optimization problem, which cannot be solved by applying efficient optimal algorithms (see Taha (1997)).

3.4.3 The problem's solution

We suggest to solve the general problem (3.17-3.23) as follows. Two hierarchical optimization levels (cycles) are imbedded in the model. At the external upper cycle the problem (call it henceforth Problem I) is as follows:

Determine optimal values $S, \{R_q\}, 1 \leq q \leq m$, to minimize the average project's non-operational conditional costs subject to the chance constraint

$$\min_{S,\{R_q\}} \{\bar{C}^{opt}|S, \{R_q\} + \{\beta(\bar{p} - p)\} \cdot K\} \tag{3.25}$$

and subject to restrictions

$$R_{q\,min} \leq R_q \leq R_{q\,max}, 1 \leq q \leq m, \tag{3.26}$$

$$S_{min} \leq S \leq S_{max}. \tag{3.27}$$

Here:

- $C^{opt}[S|\{R_q\}]$ is calculated via simulation in order to obtain a representative statistics and by solving the internal optimization problem PII (see below);
- $\bar{p}$ is the simulated statistical frequency to meet the project's due date on time, i.e., to satisfy $F < D$;

- K is a very large number (in the course of experimentation we took it to be equal to 10^{17});
- $\beta(x)$ is a zero-one function

$$\beta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.28}$$

Thus, objective (3.25) automatically prohibits cases $\bar{p} < p$, i.e., honors the chance constraint (3.18).

To solve problem PI, we use a cyclic coordinate descent algorithm which minimizes (3.25) cyclically with respect to coordinate variables S, $\{R_q\}$. Value S is optimized first, then R_1 with fixed new (optimized) S, and so forth through R_m (honoring (3.26) and (3.27). The process is then repeated starting with S again (second iteration) until the relative difference between two adjacent iterations becomes less then the pregiven tolerance $\epsilon > 0$. Thus, realizing the algorithm results in undertaking a search in a $(m+1)$-dimensional space which is a combination of values S and $\{R_q\}, 1 \leq q \leq m$, subject to restrictions (3.26-3.27). After obtaining a routine search point $(S, R_1, \ldots R_m) = X$, the internal optimization problem PII at the lower level has to be applied. Thus, values $S, \{R_q\}, 1 \leq q \leq m$, are input values for problem PII.

Problem PII is, in essence, a non-essential modification of the general problem (see Gonik (1999)). The problem boils down to determine the quasi-optimum resource delivery schedule $T(i_{\xi_A}, j_{\xi_A})$, $1 \leq \xi \leq h_A$, in order to minimize the average project duration by means of solving the resource constrained project scheduling problem via the knapsack resource reallocation problem. The general idea of the problem is as follows:

Given the due date D, the starting moment S of the project's realization and the resource levels $\{R_q\}, 1 \leq q \leq m$, determine resource delivery schedule $T(i_{\xi_A}, j_{\xi_A})$, in order to minimize the project's duration by reallocating B-resources among the project activities. Thus, the problem is as follows:

$$\min_{\{S_{ij}\},\{T(i_{\xi_A}, j_{\xi_A})\}} \{\bar{C}|\{T(i_{\xi_A}, j_{\xi_A}), S, \{R_q\}\} + [\beta(t - t^*)] \cdot K\} \tag{3.29}$$

subject to (3.17-15.1).

The model (see Sitniakovski (2002)) considers the detailed description of the heuristic algorithm for a modified version of problem (3.18-15.1, 3.29). The algorithm is, in essence, a unification of a coordinate search subalgorithm to develop a deterministic A-resource delivery schedule (see Gonik (1999), Sitniakovski (2002)) and a heuristic B-resource reallocation subalgorithm based on numerous applications of the knapsack resource reallocation problem in order to diminish the average project's duration.

The combination of $1 + h_A + m$ optimized variables $S, \{R_q\}, T(i_{\xi_A}, j_{\xi_A})$, which results in the minimal average value of the non-operational project's costs $\bar{C}|S, \{R_q\}, T(i_{\xi_A}, j_{\xi_A})$, has to be taken as the solution of the general problem (3.17-3.23). After determining beforehand (i.e., before the project starts at moment S) all optimized variables, the project has to be monitored with fixed and hired B-resources $\{R_q\}, 1 \leq q \leq m$, and with the A-resource delivery schedule $T(i_{\xi_A}, j_{\xi_A})$. Such a methodological approach can be used both for monitoring real-time projects and by undertaking experimentation via simulation in order to testify the efficiency of the general problem's solution.

Note that if solving problems PI and PII results in realizing, in the average, M_1 and M_2 search steps, correspondingly, and obtaining representative statistics to calculate $\bar{C}$ results in undertaking M_3 simulation runs to monitor the project, then determining optimized parameters $S, \{R_q\}, T(i_{\xi_A}, j_{\xi_A})$, requires in the average $M_1 \cdot M_2 \cdot M_3$ simulation runs. Thus, we recommend to apply the newly developed control model for small- and medium-size network projects. In the case of large projects we suggest to reduce the amount of the project's activities via aggregation.

3.4.4 Monitoring stochastic network projects via resource reallocation simulation model

Values $S, \{R_q\}$, and $T(i_{\xi_A}, j_{\xi_A})$, obtained by solving problems PI and PII, serve as the income parameters for the simulation model at the lower level.

The simulation model (see Golenko-Ginzburg and Gonik (1997), Golenko-Ginzburg and Gonik (1998)) comprises two submodels:

- the knapsack resource constrained reallocation to allocate B-resources among the project activities at decision points and to simulate the project's realization;

- the submodel to simulate the project's realization.

The knapsack resource reallocation problem is realized at the so-called *decision points t* when at least one activity $(i_{\omega_B}, j_{\omega_B})$ utilizing B-resources is ready to be operated but the available amount of resources is limited. A competition among the activities has to be carried out in order to chose those activities which can be supplied with resources and which have to be operated first. Assume that at a certain moment t, $\gamma < h_B$ activities $(i_{\omega_B}, j_{\omega_B}), 1 \leq \omega \leq \gamma$, are ready to be processed, but at least for one type of resource there is a lack of available B-resources.

In case of *fixed* B-resource capacities $r_{i_{\omega_B}, j_{\omega_B}, q}$, we suggest (see Golenko-Ginzburg and Gonik (1997)) to solve the zero-one integer programming problem by determining zero-one integer values ρ_ω, to maximize the objective

$$\max_{\{\rho_\omega\}} \left\{ \sum_{\omega=1}^{\gamma} [\rho_\omega \cdot p(i_{\omega_B}, j_{\omega_B}) \cdot \mu(i_{\omega_B}, j_{\omega_B})] \right\} \tag{3.30}$$

s.t.

$$\sum_{\omega=1}^{\gamma} \left[\rho_\omega \cdot r_{i_{\omega_B} j_{\omega_B} q} \right] \leq R_q(t), 1 \leq q \leq m, \tag{3.31}$$

where $p(i_{\omega_B}, j_{\omega_B})$ is the probability for activity $(i_{\omega_B}, j_{\omega_B})$ to be on the critical path in the course of a simulation run, and

$$\rho_\omega = \begin{cases} 1 & \text{if activity} (i_{\omega_B}, j_{\omega_B}) \text{ is supplied with resources} \\ 0 & \text{otherwise.} \end{cases} \tag{3.32}$$

Thus, product $W_\omega = p(i_{\omega_B}, j_{\omega_B}) \cdot \mu(i_{\omega_B}, j_{\omega_B})$ is the value activity $(i_{\omega_B}, j_{\omega_B})$ contributes to the expected project's duration. The subset of activities which being supplied with resources, results in *minimizing* the project's duration, has to be chosen. All activities entering that subset start operating at moment t. Problem (3.30-3.32) is solved by a zero-one integer programming algorithm with a precise solution. Values $p(i_{\omega_B}, j_{\omega_B})$ are obtained via simulation, by taking into account values $T(i_{\omega_B}, j_{\omega_B})$.

The simulation submodel:

1 determines decision points t to reallocate B-resources;

2 singles out activities which are ready to be processed;

3 reallocates B-resources among activities by solving the knapsack resource reallocation problem (3.30-3.32);

4 supplies activities with resources;

5 simulates the actual time durations for activities which have been supplied with A-or B-resources;

6 returns the utilized B-resources to the project's store at the moment an activity was completed;

7 calculates values W_ω for the knapsack reallocation problem at decision point t;

8 determines for all activities (i, j) their starting moments S_{ij} by using (3.24).

3.5 Conclusions

The following conclusions can be drawn from the study based on various numerical examples (see Gonik (1999), Sitniakovski (2002)):

1 The developed resource supportability model can be used in project management as a decision support model for planning and monitoring several stochastic network projects. The model has been successfully used for small and medium size projects of PERT type.

2 The developed optimal planning parameters $\{E_l, R_q\}$ (see 3.5) result in minimizing the resource average expenses for hiring and maintaining non-consumable resources. For a medium size network project with random activity durations, two cycle iterations resulted in a decrease of more than 50% in the initiated average expenses and were enough to realize the optimization process.

3 The developed resource supportability model is suitable for resource scheduling in stochastic network projects, when processing certain activities is based on delivering resources, e.g. in high technology projects, defense related industries, opto-electronics, aerospace, etc.

4 Two resource delivery models are imbedded in the project scheduling:

   - for extremely costly and rare resources the corresponding resource delivery moments have to be predetermined and calculated beforehand, i.e. before the project actually starts;
   - for limited resources which are at the disposal of the project and have to be hired at a predetermined time.

5 The objective of the resource delivery model are the total average expenses of the resource consumption within the planning horizon. Minimizing the objective is solved via simulation, in combination with a cyclic coordinate descent method and a knapsack resource reallocation model. The developed algorithm performs well and enables the model's flexibility.

6 The problems under consideration, as far as we are concerned, have not been solved yet by using other approaches. This is why it is impossible to undertake any computational comparison regarding the efficiency of the proposed heuristic models. However, it can be well-recognized that the scheduling problems comprise two sequential models:

   Model 1 centers on determining basic optimal parameters E, R_q, $T(i_{\xi_A}, j_{\xi_B})$, by means of a lookover algorithm which is a NP-complete problem (see Taha (1997), Garey and Johnson (1979)). Due to the high number of combinations we reduce the computational amount by means of implementing a heuristic coordinate descent method.

The obtained parameters serve as input values for *Model* 2 which is actually a decision-making simulation model. The latter comprises either classical knapsack or zero-one integer programming models in combination with heuristic decision-making rules. The computational time of the model's algorithm depends on the number of simulation runs.

Acknowledgement

This research has been partially supported by the Paul Ivanier Center on Robotics and Production Management, Ben-Gurion University of the Negev.

References

Bell, C. E. and Han, J. (1991). A new heuristic solution method in resource constrained project scheduling, *Naval Research Logistics* 38(3):315–331.

Garey, M. R. and Johnson, D.C. (1979). *Computer and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Co., San-Francisco.

Golenko, D. (1972). *Statistische Methoden der Netzplantechnik*, BSB B.G. Teubner Verlagsgesellschaft, Leipzig.

Golenko-Ginzburg, D. and Gonik, A. (1997). Resource constrained project scheduling with non-consumable limited resources, *International Journal of Production Economics*, 48:29–37.

Golenko-Ginzburg, D. and Gonik, A. (1998). A heuristic for network project scheduling with random activity durations depending on resource reallocation, *International Journal of Production Economics*, 55:149–162.

Golenko-Ginzburg, D., Gonik, A. and Sitniakovski, Sh. (2000). Resource supportability model for stochastic network projects under a chance constraint, *International Journal of Communication in Dependability and Quality Management* 3(1):89–102.

Golenko-Ginzburg, D., Gonik, A. and Sitniakovski, Sh. (2000). Resource constrained project scheduling for several stochastic network projects, *International Journal of Communication in Dependability and Quality Management* 3(1):63–73.

Golenko-Ginzburg, D., Ljubkin, S., Rezer, V. and Sitniakovski, Sh. (2001). Algorithms of optimal supply of resources to a group of projects, *International Journal of Automation and Remote Control* 62(6):1366–1375.

Gonik A. (1995). Planning and controlling multilevel stochastic projects, Ph. D. Thesis, Ben-Gurion University of the Negev, Beer-Sheva, Israel.

Gonik A. (1999). Resource scheduling model with cost objective for stochastic network projects, *International Journal of Communication in Dependability and Quality Management* 2(1):102–108.

Kolish R. (1995). *Project Scheduling Under Resource Constraint*, Physica-Verlag.

Luss H. (1991). A non-linear minimax allocation problem with multiple knapsack constraints, *Operations Research Letters* 10(4):183–187.

Luenberger D. G. (1973). *Introduction to Linear and Non-Linear Programming*, Addison-Wesley Publishing Co., Reading, Massachusetts.

Sitniakovski Sh. (2002). Control and scheduling models in stochastic project management, Ph.D. Thesis, Ben-Gurion University of the Negev, Beer-Sheva, Israel.

Taha H. A. (1997). *Operations Research, An Introduction*, Collier Macmillan International Editions, New York, London.

Toker, A., Kondacsi, S. and Erkip, N. (1991). Scheduling under a non-renewable resource constraint, *Journal of the Operational Research Society* 42(9):811–814.

Voropajev, V., Ljubkin, S., Golenko-Ginzburg, D. and Gonik, A. (1999). A model for supplying with constrained resources in project management under random disturbances, *Project Management* (International Project Management Journal) 5(1):68–73.

Zhan J. (1994). Heuristics for scheduling resource constrained projects in MPM networks, *European Journal of Operational Research* 76(1):192–205.

Chapter 4

DUE DATES AND RCPSP

Francisco Ballestín[1], Vicente Valls[2], Sacramento Quintanilla [3]

[1]*Dpto. de Estadística e Investigación Operativa, Facultad de Ciencias Económicas y Empresariales, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006, Pamplona, Spain.*

Francisco.Ballestin@unavarra.es

[2]*Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot, Valencia, Spain.*

Vicente.Valls@uv.es

[3]*Dpto. de Economía Financiera y Matemática, Facultad de Económicas y Empresariales, Universitat de Valencia, Avda. de los Naranjos, s/n, Edificio Departamental Oriental, Valencia, Spain.*

Maria.Quintanilla@uv.es

Abstract Due dates are an essential feature of real projects, but little effort has been made in studying the RCPSP with due dates in the activities. This paper tries to bridge this gap by studying two problems: the TardinessRCPSP, in which the objective is total tardiness minimization and the DeadlineRCPSP, in which the due dates are strict (deadlines) and the objective is makespan minimization. The first problem is NP-hard and the second is much harder, since finding a feasible solution is already NP-hard. This paper has three objectives: Firstly to compare the performance on both problems of well-known RCPSP heuristics - priority rules, sampling procedures and metaheuristics - with new versions we have developed that take due dates into consideration. Secondly, to present an instance generator that can generate instances with loose, medium, and tight due dates for computational study. And, finally, to adapt the technique of justification to deal with due dates and deadlines and to show its profitability.

Keywords: Project management; due dates; heuristics.

4.1 The RCPSP with due dates

The objective of the classical Resource Constrained Project Scheduling Problem is to schedule the set of activities of a project in the minimum time and is subjected to two types of restrictions. Firstly, some activities must end before others can start due to the precedence relations. Secondly, processing each activity requires several types of resources, which are available at a constant and fixed rate throughout the project duration. Nevertheless, the emergence of new restrictions in the practice forces the extension of the classical model. One of the most common additional restrictions is that some activities must end before a given due date.

Due dates in project scheduling first appear in the literature in Vanhoucke et al (1999), where the WETPSP (weighted earliness-tardiness project scheduling problem) is solved. In this problem, only precedence restrictions are considered and the objective is to finish activities as near as possible to their due dates. There is a cost associated with both the earliness (an activity finishes before its due date) and the tardiness (an activity finishes after its due date) of each activity. In a later paper, Vanhoucke et al (2001) work with a generalisation of the WETPSP, the RCPSPWET (resource-constrained project scheduling problem with weighted earliness-tardiness costs), in which resource restrictions are also present.

The interest for problems involving due dates in the field of machine scheduling dates back to 1955, in a paper by Jackson. Gordon et al (2002) survey the literature concerning the models involving machine scheduling and due dates. The large number of referenced papers shows that the machine scheduling problems including due dates are of permanent interest.

This paper deals with RCPSP and due dates and deals with two different problems. In the first problem (TardinessRCPSP) the objective is total tardiness minimization. In the second problem (DeadlineRCPSP) the due dates are strict (deadlines) and the objective is makespan minimization. The later problem is much harder, since finding a feasible solution is already NP-hard (Garey and Johnson (1979).

Koulamas (1994) provides a unified framework for the total tardiness problem by surveying the related literature in machine scheduling.

One of the ways proposed in the literature in which deadlines can be modelled, namely by using maximum time lags. A maximum time lag δ between the beginning of activities i and j means that j must begin at most δ units of time after the beginning of i. Therefore, a maximum time lag between the source and the end of activity i is equivalent to imposing a deadline in this activity. Neumann et al (2003) describe several procedures for the RCPSP with time lags that can be applied to the DeadlineRCPSP. However, they cannot be used to solve the TardinessRCPSP.

TardinessRCPSP and DeadlineRCPSP are denoted by $PS \mid prec \mid T$ and $PS \mid temp \mid C_{max}$ respectively in the notation of Brucker et al (1999) or $m, 1|cpm|T$ and $m, 1|cpm, \delta_j \mid C_{max}$ in the notation of Herroelen et al (1998). Being

$$T = \sum_{j=1}^{n} \max(0, s_j + d_j - dd_j)$$

where n is the number of activities and s_j, d_j and dd_j the starting time, duration, and due date of activity j, respectively.

The main objective of this paper is to compare the performance on both problems of well-known heuristics (priority rules, sampling procedures and metaheuristics) developed for the RCPSP with new versions we have developed that take due dates into consideration. The computational study is carried out on instances we have generated by assigning due dates to the instances in the standard set j120 (Kolisch et al (1995)) in such a way that instances with loose, medium, and tight due dates are obtained. Section 16.2 describes the new procedure developed to assign due dates in these instances. Sections 16.3 and 16.4 deal with the TardinessRCPSP and DeadlineRCPSP, respectively. First of all, the performance of priority rules, sampling procedures and metaheuristics developed for the RCPSP is compared with new versions of the algorithms that take due dates into account. We have also adapted the technique of justification and studied whether the efficiency of this procedure (Valls et al (2005a)) is applicable to the newly studied problems. The results for the DeadlineRCPSP have been compared with those obtained by a state-of-the-art heuristic algorithm for the RCPSP with maximum time lags. Finally, a summary and some concluding remarks appear in Section 16.5.

4.2 Generation of instances

Our purpose is to generate test instances by introducing due dates in the standard set j120 for the RCPSP generated using ProGen (Kolisch et al (1995)). The set j120 consists of 600 projects with four resource types and 120 activities. These instances were generated under a full factorial experimental design with the following three independent problem parameters: network complexity, resource factor, and resource strength. Details of these problem instances are given in Kolisch et al (1995) and Kolisch and Sprecher (1997).

Vanhoucke et al (2001) assign due dates to projects to use them as test instances for the RCPSPWET. The due dates are generated as follows: first, they obtain a maximum due date for the project by multiplying the critical path length by a factor from the set {1, 1.25, 1.5, 1.75, 2, 2.25 and 2.5}. Then they randomly generate numbers between 1 and the maximum due date. Then the activities are ordered topologically. Finally, they sort these numbers and assign them to the activities in increasing order, i.e., activity 1 has the lowest due date, activity 2 the second lowest, etc. In a later work, Vanhoucke (2002) studies how to assign due dates to a particular project, basing the assignment on negotiation arguments between contractor and client.

Neumann et al (2003) assign due dates randomly, according to a uniform distribution in

$$[0, \lceil 1.5 \times ES_n \rceil],$$

where ES_i is the earliest starting time of i. Problem instances are generated in a computational study of a branch and bound algorithm for the RCPSPWET with maximum time lags.

Our approach is different. We want to differentiate between:

1 Instances with loose due dates, when it is easy to fulfil most of the due dates at the same time and only a limited subset of activities are going to be tardy in any schedule.

2 Instances with tight due dates, when many activities are going to be tardy in any schedule.

3 Instances in between, which we will put in the so-called *medium set.*

Therefore we want to create three sets, *loose, medium and tight.* We will see later that some algorithms offer a different behaviour in these sets, thus supporting the partition we have made.

Given an instance I in the j120 set the due date assignment is made in such a way that Tardiness(S(I)) belongs to a prefixed interval $[a, b]$. $S(I)$ is the sequence generated by applying the parallel schedule generation scheme with the due date priority rule (the shorter the due date the larger the priority value). A formal description of the parallel schedule generation scheme can be found, e.g., in Kolisch et al (1995). For simplicity Tardiness(S(I)) will be denoted by Tardiness(I).

After a preliminary study with instances belonging to j120 we have associated the following intervals with each of the sets. If Tardiness(I) belongs to the interval [1, 199], [200, 999] or [1000, 9999] the instance I belongs to the loose, medium or tight set, respectively.

An instance where Tardiness(I) is equal to 0 or more than 10000 is considered to fit in other sets which we could call *very loose* and *very tight* respectively. In the first case, the resulting instances are too easy for the algorithms considered in this paper. On the other hand, no due date assignment can transform many j120 instances into instances in the very tight set. Furthermore, we believe that the inclusion of the very tight instances in the test set would not add further insights into the computational analysis.

Given a j120 instance I, the algorithm Generate_Instance$(I, a, b, \epsilon, nsol)$ generates an instance with Tardiness(I) in the interval $[a, b]$ where ϵ and $nsol$ are parameters that will be introduced later. For simplicity, the generated instance will be also denoted by I. The algorithm makes calls to the function

Due_Dates(I, α, β, pc), which assigns due dates to the activities in the following manner: to a percentage pc of the activities the sum of their earliest finishing time EF plus a random integer number in $[a, b]$ and to the rest of activities their EF. Figure 16.1 shows the algorithmic scheme of the Due_Dates(I, α, β, pc) function.

Figure 16.2 shows the algorithmic scheme of Generate_Instance$(I, a, b, \epsilon, nsol)$. Step 1 ends the procedure if the impossibility of generating an instance with tardiness in $[a, b]$ is discovered. Step 2 checks whether the procedure is able to generate instances with tardiness greater than a. If not, the instance Due_Dates $(I, 0, 0, 1)$ is returned.

In step 3, first the variable end is fixed as the minimum integer number for which the assignment $dd_i = EF_i + end * \epsilon * CP$ results in Tardiness(I) = 0. Then, the procedure repeatedly calls the function Due_Dates to compute due dates for the 100% of the activities and considering all the intervals included in the interval $[0, end * \epsilon * CP]$ with extremes multiples of $\epsilon * CP$. This variety of intervals leads to a variety of instances all with tardiness in the interval $[a, b]$. In some instances, the due dates of all activities are similarly tight, or loose; in others, the due dates are much tighter for some activities than for others.

If step 3 fails to generate an appropriate instance, then step 4 tries to generate one by still computing due dates for all activities and considering intervals $[0, h]$ with decreasing values of h, starting with $h = \epsilon * CP$.

If step 4 also fails, then step 5 tries to generate an appropriate instance by setting $dd_i = EF_i + 1$ to decreasing percentages of activities.

We have applied this procedure to the 600 instances in j120 with $nsol = 10$ and $\epsilon = 0.05$ and the intervals [1, 199], [200, 999], and [1000, 9999] thus obtaining 600, 600, and 470 instances in the loose, medium, and tight sets, respectively. For the other 130 instances, Tardiness(Due_Dates (I,0,0,1)) < 1000 so they cannot lead to instances in the tight set.

4.3 The TARDINESSRCPSP

4.3.1 Priority rules

The first attempts to heuristically solve the RCPSP were done by using priority rules to decide the order in which activities are considered for the allocation of scarce resources.

Since then, new and more elaborate priority rules have been proposed even when higher quality algorithms had been proposed (see Kolisch (1996), Özdamar and Ulusoy (1996), Thomas and Salhi (1997)). The reason is that they are still believed to be important (see Kolisch and Hartmann (1999)), since they can provide good initial solutions for other algorithms at a low computational cost.

```
for ($i$ = 1 ; $i \leq n$; $i$ = $i$+1)
{
    Let $p$ be a random number in [0,1].
    Let $q$ be a random integer number in $[\alpha, \beta]$.
    if $p \leq pc$
      set $dd_i = EF_i + q$
    else
      set $dd_i = EF_i$
}
let $I$ be the instance after assigning the due dates
return $I$
```

Figure 4.1. Due_Dates(I, α, β, pc) function.

Priority-rule based scheduling is made up of two components, a schedule generation scheme (SGS) and a priority rule. The basic SGSs are the Serial SGS (S-SGS) and the Parallel SGS (P-SGS). S-SGS yields active schedules and P-SGS yields non-delay schedules (see Kolisch et al (1995)).

According to Kolisch et al (1995), who made experiments in j30, the best priority rule for the S-SGS is the LFT (Latest Finishing Time) rule, while WCS (Worst Case Slack) is the best for the P-PGS. MINSLK is another popular priority rule. We are going to observe the performance of these rules in the TardinessRCPSP and compare it to that of their 'natural' adaptations to the TardinessRCPSP, obtained by just changing LF to dd.

The adapted rules are:

1 The EDD rule, which schedules first the activities with the earliest due date; this rule can be applied with the S-SGS and P-SGS. This rule can be seen as an adaptation of the LFT rule and it is a well-known priority rule in machine scheduling.

2 The MINSLK_dd rule. In each iteration of the S-SGS, the activity i with minimun slack $dd_i - EST_i$ is scheduled. EST_i is the earliest starting time of i if i is scheduled in this iteration.

3 The WCS_dd rule, which is an adaptation of the WCS rule. WCS is used only with the P-SGS. At each schedule time t, it selects the activity i with minimum $LF_i - d_i - \max\{E(j,i)/i, j \text{ eligible at } t\}$. LF_i is calculated as in the rule LFT, with backward recursion and the critical path length as the latest finishing time of the entire project. $E(j,i)$ is the earliest

```
1 If (Tardiness(Due_Dates ($I$,0,0,1)) $< a$) no instance can be generated. STOP.

2 If(Tardiness(Due_Dates ($I$,0,0,1)) = $a$) store $I$. Go to 6.

3 $end = 1$.
  while (Tardiness(Due_Dates $(I, end * \epsilon, end * \epsilon, 1)) \neq 0)$ $end++$
  for($i = 0$; $i < end$; $i++$)
  for($j = i + 1$; $j \leq end$; $j++$)
      for($k = 0$ ; $k < nsol$ ; $k++$)
          if(Tardiness(Due_Dates $(I, i * \epsilon * CP, j * \epsilon * CP, 1)) \in [a, b]$),
          store $I$
  If at least one instance has been stored, go to 6.

4 for($j = 0$; $\epsilon * CP - j > 0$;$j++$)
  for ($k = 0$; $k < nsol$; $k++$)
      if(Tardiness(Due_Dates $(I, 0, \epsilon * CP - j, 1)) \in [a, b]$), store $I$.
      Go to 6.

5 for($percentage = 0.9$; $percentage \geq 0$; $percentage = percentage - 0.1$)
  for ($k = 0$ ; $k < 10 * nsol$; $k++$)
      if(Tardiness (Due_Dates $(I, 1, 1, percentage)) \in [a, b]$),
      store $I$. Go to 6.

6 Choose one of the stored instances at random and return it. STOP.
```

Figure 4.2. Generate_Instance$(I, a, b, \epsilon, nsol)$.

starting time to schedule activity i if j is started at t. The WCS_dd rule uses dd_i instead of LF_i.

Table 16.1 shows the results. The first column indicates the rule and the SGS used (P: parallel and S: serial). The three remaining columns show the average total tardiness for each of the three considered sets. The best result obtained in each set is shown in bold. The position of each procedure in the relative ranking for each set is shown in parenthesis.

Table 4.1. Average total tardiness and ranking with priority rules.

	loose set		medium set		tight set	
LFT+S	367	(6)	802	(5)	2060	(5)
LFT+P	182	(3)	512	(2)	**1633**	(1)
WCS+P	208	(4)	556	(4)	1723	(2)
EDD+S	246	(5)	808	(6)	2224	(6)
EDD+P	**73**	(1)	**498**	(1)	1742	(3)
MINSLK_dd+S	663	(7)	1269	(7)	2803	(7)
WCS_dd+P	87	(2)	520	(3)	1810	(4)

It seems that, as in the RCPSP, the Parallel SGS behaves better than the Serial SGS, because LFT and EDD are better with Parallel. Besides, the two worst (best) priority rules in every set use the Serial (Parallel) SGS.

The best rules in the loose set are EDD+P and WCS_dd+P, the best in the medium set are WCS_dd+P and LFT+P, whereas the best rules in the tight set are LFT+P and WCS+P.

Concerning the use of the Serial SGS (LFT+S, dd+S and MINSLK_dd+S), the adapted rule EDD is the best in the loose set, whereas LFT+S is the best in the medium and tight sets.

It seems that the best adapted rules outperform the RCPSP rules in the loose set, and are at least as good as those in the medium set. However, in the tight set the best RCPSP rules are better than the adapted ones. This may be so because the RCPSP rules find an early beginning for all the activities, since they are good in minimising the makespan. Adapted rules, such as EDD+P and WCS_dd+P, seem to be better in finding a very early beginning for a subset of activities. Therefore they are better in the first two sets, where only a subset of activities have problems with their due dates.

We can see that some algorithms perform differently depending on the set. For instance, there is a big difference between WCS_dd+P and LFT+P, in favour of the former in the loose set and against it in the tight set. This behaviour of the algorithms supports the partition of instances we have made. The differences among sets observable in the following sections also corroborate the partition, but we will not repeat the argument again.

4.3.2 Sampling procedures

Sampling methods are natural generalisations of priority rules. They generally use one priority rule and one SGS, and calculate different schedules by biasing the selection of the priority rules through a random device. We are going to use the regret-based biased random sampling, which is the best RCPSP sampling method (see Kolisch et al (1995), Schirmer and Riesenberg (1997)). Specifically, we use the best five priority rules according to Table 16.1: LFT+S, LFT+P, WCS+P, EDD+P and WCS_dd+P. B+priority rule denotes the procedure regret-based biased random sampling with the priority rule.

We are also going to run the procedures Random+S and Random+P. They simply randomly generate activity lists and then schedule them with the S-SGS or the P-SGS, respectively. The output of the algorithm is the best schedule obtained. With these algorithms we can analyse the quality of active and non-delay schedules. We may also use them to assess the quality of the other algorithms according to how much the other algorithms outperform them. All algorithms will be run until 5000 schedules are generated. This is usually the upper limit imposed when comparing state-of-the-art heuristic algorithms for the RCPSP. Table 16.2 shows the results.

Table 4.2. Average total tardiness and ranking with sampling procedures.

	loose set		medium set		tight set	
Random+S	101	(6)	493	(7)	1723	(7)
Random+P	28	(3)	305	(5)	1385	(4)
B+LFT+S	104	(7)	446	(6)	1598	(6)
B+LFT+P	33	(4)	287	(2)	**1321**	(1)
B+WCS+P	35	(5)	298	(4)	1353	(2)
B+EDD+P	**17**	(1)	**282**	(1)	1368	(3)
B+WCS_dd+P	18	(2)	292	(3)	1398	(5)

As happens in the RCPSP (see Kolisch et al (1995)), priority rules which perform good for single-pass approaches do so for biased random sampling approaches and vice versa. Besides, we obtain approximately the same ranking as for the single-pass case. Again, the P-SGS performs much better than the S-SGS. This is also true in the case of the RCPSP, at least in j120 (see Kolisch and Hartmann (1999)). One of the most astonishing aspects of the table is the behaviour of Random+P, especially in the loose set. It outperforms all the other samplings but two, B+EDD+P and B+WCS_dd+P. In the other two sets it is worse than most of the rest of algorithms, but the difference is not as large as it was expected.

4.3.3 Metaheuristic procedures

In recent years, as in many other optimisation problems, several metaheuristic algorithms have been developed for the RCPSP, having outperformed other types of algorithms. One of the most important of these algorithms is the activity list genetic algorithm of Hartmann (1998), which has been the best heuristic algorithm for the RCPSP, until recently. It outperforms, among others, the best sampling procedures (see Kolisch and Hartmann (1999)). We have programmed this algorithm and run it with a limit of 5000 schedules. We have changed only the evaluation of the fitness of a solution; in the original algorithm, it was the makespan, now, it is the total tardiness. The results can be seen in the third line of Table 16.3. The second line of the table contains the best results so far, those obtained by the sampling procedures.

Table 4.3. Average total tardiness.

	loose set	medium set	tight set
best results so far	**17**	282	1321
Hartmann	30	**272**	**1292**

As was predictable, the Hartmann algorithm obtains better results in the medium and tight sets. However, the difference is not as great as it could be expected. Besides, its results in the loose set are worse than the best three samplings (B+EDD+P, B+WCS_dd+P and Random+P). Taking everything into account, it seems that the performance of the Hartmann algorithm is not as good as in the RCPSP. One of the reasons for the bad performance in the loose set seems to be the initial solutions, which are obtained with B+LFT+S. As we have seen before, the non-delayed schedules are much better on average that the active schedules. Bearing this in mind, we have changed the procedure to generate the initial population in the Hartmann algorithm, allowing it to employ B+EDD+P to obtain initial solutions. Table 16.4 contains the results obtained. The percentages of columns 1–2 refer to the % of solutions calculated by each procedure written at the head of these columns.

Table 4.4. Average total tardiness with Hartmann algorithm changing the initial solutions.

B+LFT+S	B+EDD+P	loose set	medium set	tight set
100%	0%	30	272	1292
0%	100%	**11**	245	1297
50%	50%	**11**	**238**	**1271**

The best algorithm, which we will call Hartmann(2) from now on, is the one that calculates the 50% of the solutions of the initial population with B+EDD+P and the other 50% with B+LFT+S. Note that the number of solutions that this algorithm calculates with the P-SGS is 50, whereas the samplings with this SGS calculate 5000. So, the genetic algorithm is capable of taking advantage of these few non-delay solutions - and the active solutions calculated with LFT - to finally provide a better solution. Hartmann(2) improves the best heuristic averages by 54.5%, 18.5% and 3.93% in the loose, medium and tight set respectively.

4.3.4 The justification

The justification (by extremes) of a schedule S to the right (left) consists of scheduling every activity as late (early) as possible in decreasing (increasing) order of their ends (beginnings) without changing the schedule period of the other activities. This technique was introduced in 1964 by Wiest and has not been used to its full potential until recently. It is a very easy and fast procedure that never increases the makespan of a schedule when applied to it and in many cases shortens it. It has been used, among others, by Li and Willis (1992), and Tormos and Lova (2001). Valls et al (2005a) have proved that it can be easily incorporated into a wide range of algorithms for the RCPSP, increasing the solution quality and maintaining the number of schedules calculated. The justification was able to improve well-known priority rule based procedures and random sampling methods, as well as to transform several middle quality heuristic algorithms into algorithms that outperform state-of-the-art heuristic algorithms for the RCPSP.

We want to check whether the justification (by extremes) can also improve algorithms for the TardinessRPCPS. We have added it to the algorithms we have seen until now, so that the double justification (DJ) is applied to every schedule calculated by the original algorithm. The DJ of a schedule S consists of, firstly, justifying S to the right, and then justifying the resulting schedule to the left. We still impose the limit of 5000 schedules in every algorithm - 1666 schedules generated by the original algorithm and 3332 schedules generated by DJ - except in the case of the single-pass algorithms. The DJ versions of these algorithms calculate three schedules instead of one. It is important to remark that the application of DJ to a schedule in the TardinessRCPSP can worsen its evaluation, contrary to what happens in the RCPSP. As an example, Figure 4.3 shows a project and two feasible schedules S and S'. There is only one type of resource ($K = 1$) and r_{i1} is the quantity of the resource required by the activity i. S' is the result of double justifying S. If activity 6 has assigned a due date equal to 4, the total tardiness is 0 before justifying and 1 after DJ. The makespan is one unit less after DJ. This is the first application of the justification to a problem with an objective function different from the makespan.

Table 16.5 contains the results of the best algorithms we have seen up to now, with and without DJ, as well as the improvement percentage that DJ produces. This percentage is calculated as

$$\text{Average total tardiness} = \frac{\text{average(algorithm)} - \text{average(algorithm+DJ)}}{\text{average(algorithm + DJ)}}$$

DJ improves all the single-pass algorithms in all the sets. These improvements are large in some cases, even in cases of good priority rules. DJ greatly improves the sampling methods that use the S-SGS, which are significantly

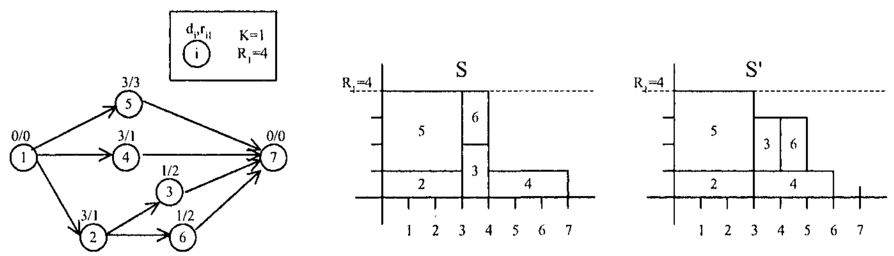


Figure 4.3. Schedules before and after DJ

Table 4.5. Average total tardiness with and without double justification and the improvement.

	loose set			medium set			tight set		
		+DJ	imp.		+DJ	imp.		+DJ	imp.
LFT+S	367	311	18%	802	715	12%	2060	1928	7%
LFT+P	182	170	7%	512	488	5%	1633	1592	3%
WCS+P	208	194	7%	556	531	5%	1723	1680	3%
EDD+P	73	61	20%	498	423	18%	1742	1612	8%
WCS_dd+P	87	69	26%	520	448	16%	1810	1677	8%
Random+S	101	70	44%	493	379	30%	1723	1512	14%
Random+P	28	30	-7%	305	298	2%	1385	1371	1%
B+LFT+S	104	89	17%	446	389	15%	1598	1482	8%
B+LFT+P	33	34	-3%	287	285	1%	1321	1311	1%
B+WCS+P	35	36	-3%	298	294	1%	1353	1342	1%
B+EDD+P	17	18	-6%	282	272	4%	1368	1332	3%
B+WCS_dd+P	18	19	-5%	292	281	4%	1398	1361	3%
Hartmann	30	31	-3%	272	240	13%	1292	1178	10%
Hartmann(2)	**11**	12	-8%	238	**213**	12%	1271	**1179**	8%

worse than those that use P-SGS. These good sampling procedures are not significantly affected positively by DJ. In fact, B+EDD+P and B+LFT+P perform better without DJ, since the losses in the loose set are bigger than the amount of improvement in the other sets. It could be thought that DJ is only useful for the most basic algorithms, but this is not the case. Although DJ worsen Hartmann and Hartmann(2) in the loose set, the improvements in the medium and the tight sets make up for the losses.

Several remarks should be made:

1 The results in the medium and tight set are never worse when DJ is applied. In section 3.5 we will give a possible explanation for this.

2 In the Hartmann algorithms, S is changed to DJ(S) only if DJ(S) is better. This might be an important issue because the algorithm performs differently if DJ(S) always replaces S. We have chosen the first approach because it yields better results in this case.

3 We have also calculated the percentage of solutions improved and worsened by DJ (and the amount of improvement) in each set for the random samplings. Table 10.6 offers this information.

Table 4.6. Effect of DJ on randomly generated schedules.

	loose set			medium set			tight set		
	%imps	%wor	avimp	%imps	%wor	avimp	%imps	%wor	avimp
Random+ S	68	22	258	91	8	91	99	1	18
Random+ P	22	45	77	50	46	43	56	44	6

where,
% imps = percentage of schedules improved by DJ.
% wor = percentage of schedules worsen by DJ.

$$avimp = \frac{\sum_{S \in Imp*(DJ)} \frac{f(S)-f(DJ(S))}{f(DJ(S))}}{|Imp * (DJ)|}$$

where $Imp * (DJ) = \{S/f(DJ(S)) < f(S), f(DJ(S)) \neq 0\}$ and f = Tardiness.

The effect of adding DJ to Random+S and Random+P is quite different. First of all, many Random-S schedules are improved by the DJ, whereas the percentage drops to 22-56% in the case of Random-P schedules. Besides, the average improvement among the improved solutions in the Random-S case is more than twice that of the Random-P case. Moreover, the percentage of schedules worsened by DJ is several times bigger in the latter case. Anyway, the figures for the Random-P schedules are very good, especially if we bear in mind that DJ does not improve much the Random+P procedure.

There are however similarities in the behaviour of the DJ in both sampling methods. As the difficulty of the instances increases, the percentage of improvements increases, whereas the average improvement among the improved solutions decreases.

4.3.5 Justification by eligibles

The justification by extremes has been proved to decrease the makespan of a great percentage of schedules. The justification changes the beginnings of many activities, some of them are advanced and others are postponed. However, the total amount of units that the former activities advance is bigger (in general) that the amount in which the latter are postponed. This is endorsed by an experiment carried out by Ballestín (2002). In each of the 600 instances of j120, the DJ was applied to 1666 random active schedules (so that no more than 5000 schedules were calculated in total). In every of the 600*1666 schedules, the following relative improvement was calculated,

$$\frac{\sum_{i=1}^{n}(s_i - DJ(S)_i)}{\sum_{i=1}^{n} s_i} = \frac{\sum_{i\in Adv(S)}(s_i - DJ(S)_i) - \sum_{i\in Post(S)}(DJ(S)_i - s_i)}{\sum_{i=1}^{n} s_i}.$$

where s_i is the beginning of activity i in the initial schedule S, $DJ(S)_i$ is the beginning of activity i after the DJ, Adv(S) (Post(S)) is the set of activities that are advanced (postponed) after the DJ. The average of these ratios was 6.86%.

In the medium and tight sets, there are usually many tardy activities in every schedule S. Therefore it is easy for some or many of them to belong to Adv(S). Although some of them may also belong to Post(S), the bigger cardinality of Adv(S) and the more units that Adv(S) activities are moved mean that many schedules are improved with respect to total tardiness objective function after the DJ. In fact, the percentage is so big that algorithms with DJ outperform those without it.

In the loose set, there are few tardy activities in most schedules. It is easier than in the previous cases that Adv(S) does not contain any of them. In this situation, if Post(S) contains some tardy activities, DJ(S) will have a worse total tardiness evaluation than S, no matter how positive the above ratio is. We have indeed seen that the percentage of schedules worsened by DJ in the loose set is the biggest.

It is obvious that with the justification by extremes we may be advancing and/or postponing the 'wrong' activities, i.e., advancing non-tardy activities, postponing tardy ones or changing non-tardy activities into tardy ones. Thus, we obtain a worse solution, or not as good as it could be. For these reasons, it seems clear that we may obtain better results if we adapt the justification we use for the problem at hand, changing the order in which activities are justified. For example, when we are justifying to the left, it seems sensible to justify first the tardy activities, to try to advance them and hence improve the objective function. Unfortunately, it is not as easy as that, because justifying them first might mean not finding gaps in the schedule to move them at all. Anyway, it seems interesting to study a more general type of justification than the justification by extremes, the justification by eligibles (see Valls et al (2006)). The justification by eligibles of a schedule to the right (left) consists of scheduling every activity as late (early) as possible in a specified order. In order to completely specify the justification by eligibles, we only need to specify this order, the rule to choose

the activity that is going to be justified either to the right or to the left in each iteration. The activity to be justified is selected among the so-called eligible activities. When justifying to the right (left), an activity is eligible if all its successors (predecessors) have been already justified. In all rules we are going to justify each activity once, even if later on it could be further justified. With this condition we can still count each justification of a schedule as one schedule, since in total we make n activity justifications. Therefore we can still impose the limit of 5000 schedules as a basis to compare the algorithms. We will test the following rules:
Justification to the right:

- R1: $\max\{s_i + d_i, i \text{ eligible}\}$
- R2: $\max\{dd_i, i \text{ eligible}\}$
- R3: $\max\{new\_end_i, i \text{ eligible}\}$
- R4: $\max\{s_i + d_i, i \text{ eligible and non-tardy}\}$
- R5: $\max\{new\_end_i, i \text{ eligible and non-tardy}\}$

where new_end_i is the end of activity i if it was justified in the current iteration.
Justification to the left:

- L1: $min\{s_i, i \text{ eligible}\}$
- L2: $min\{dd_i - d_i, i \text{ eligible}\}$
- L3: $min\{new\_begi, i \text{ eligible}\}$
- L4: $min\{s_i, i \text{ eligible and tardy}\}$
- L5: $min\{new\_beg_i, i \text{ eligible and tardy}\}$
- L6: $max\{imp_i, i \text{ eligible and with the minimun } new\_beg_i\}$
- L7: $max\{imp_i, i \text{ eligible}\}$

where new_beg_i is the beginning of activity i if it was justified in the current iteration and imp_i is the improvement the total tardiness function would have if i began in new_beg_i.

Notice that the justification by eligibles with the combination LR1, RR1 is equivalent to the justification by extremes.

If we combine these rules and use them in the Hartmann(2) algorithm, we obtain the results shown in Table 10.7.

The results are very rule dependent. Hartmann(2) with the combination R1-L6 is capable of outperforming the best results so far in the medium and tight sets, without worsening the results in the loose set.

Table 4.7. Average total tardiness with Hartmann(2) with justification by eligibles.

	R1			R2			R3			R4			R5		
L1	12	213	1179	12	210	1165	12	209	1162	14	227	1207	14	226	1204
L2	12	209	1162	13	241	1269	13	222	1199	12	246	1288	12	245	1290
L3	12	209	1165	11	213	1180	11	203	1126	12	214	1176	12	215	1177
L4	13	220	1192	14	254	1297	14	231	1220	15	259	1315	15	259	1307
L5	13	219	1195	14	254	1291	14	229	1214	15	259	1304	15	255	1302
L6	**11**	**199**	**1119**	12	228	1217	12	207	1145	11	226	1231	11	226	1228
L7	12	210	1166	13	237	1255	12	214	1181	13	243	1269	13	241	1271

4.4 The DEADLINERCPSP

From now on we are going to deal with the DeadlineRCPSP, the version of the problem in which due dates take the form of strict restrictions for the activities. That is, we consider the RCPSP with n added constraints, $s_i + d_i \leq dd_i$, $i = 1, \ldots, n$. Note that the objective of this problem is makespan minimization.

The most important difference between this problem and the TardinessR-CPSP is that the first one may have no feasible solutions (and even the feasibility problem is NP-hard) whereas the second one is always feasible and it is very easy to find a feasible solution to it. Therefore, one way to evaluate the quality of an algorithm for the DeadlineRCPSP is the percentage of instances for which it obtains a feasible solution. We will call this the *first measure of quality*. However, it should also be taken into account how well an algorithm minimises the makespan in the instances where it finds feasible solutions. We will call this measure the *second measure of quality*. Both measures should be taken into account to compare algorithms.

Given the unfeasibility of most instances in the medium and tight sets we have restricted the computational results to instances in the loose set. We have not considered priority rules in this study, as they rarely produce feasible solutions.

4.4.1 Sampling procedures and the Hartmann algorithm

First of all, we are going to compare the results of the sampling procedures and the Hartmann algorithms in the DeadlineRCPSP. In Table 10.8 we can observe the number of instances for which these algorithms obtain a feasible solution. The number of instances for which at least one of the algorithms considered in this paper has obtained a feasible solution is 399. The best result obtained is shown in bold. The position of each procedure in the relative ranking is shown in parenthesis.

Table 4.8. Number of feasible solutions and ranking.

	# feasible solutions (ranking)	
Random+S	55	(8)
Random+P	210	(4)
B+LFT+S	44	(9)
B+LFT+P	148	(6)
B+WCS+P	143	(7)
B+EDD+P	281	(2)
B+WCS_dd+P	280	(3)
Hartmann	187	(5)
Hartmann(2)	**322**	(1)

Table 4.9. Average deviation in percentage with respect to the best solution known.

	113	123	152	199	252	268
B+WCS+P	6.14 (3)					
B+LFT+P	6.33 (4)	6.55 (3)				
Hartmann	**4.96 (1)**	**5.01 (1)**	**5.21 (1)**			
Random+P	7.05 (7)	7.23 (6)	7.81 (5)	7.96 (4)		
B+WCS_dd+P	6.86 (6)	6.86 (4)	7.18 (4)	7.13 (3)	6.93 (3)	
B+EDD+P	6.85 (5)	6.86 (4)	7.07 (3)	6.96 (2)	6.85 (2)	6.87 (2)
Hartmann(2)	5.41 (2)	5.51 (2)	5.63 (2)	**5.73 (1)**	**5.59 (1)**	**5.55 (1)**

The worst algorithms are B+LFT+S and the Random+S algorithm, which obtain very few feasible solutions. The best algorithms are Hartmann(2), B+EDD+P and B+WCS_dd+P. These results are in accordance with those obtained in the loose set in the previous section.

To compare some algorithms with respect to the second measure we have to restrict ourselves to the instances where these algorithms obtain feasible solutions. We proceed as follows. We compare all algorithms but the two worst ones, 7 in total. We make 6 comparisons; in the first one we compare all 7, then the 6 best, then the 5 best, etc. This way we will be able to have several comparisons between most algorithms without having to compare them in pairs. We have ordered the algorithms according to the number of instances for which they obtain feasible solutions. The results are shown in Table 10.9 where the numbers in italics stand for the number of instances taken into account to make the comparisons. The results show the average deviation in percentage with respect to the best solution known for these instances. To obtain the deviation at an instance i from a particular algorithm we calculate the quotient

$$\frac{\text{alg(i)} - \text{best(i)}}{\text{best(i)}}$$

where alg(i) is the makespan obtained by this algorithm and best(i) is the best known solution. The best known solution has been obtained considering all the algorithms in this paper. The numbers in parenthesis show the ranking.

The most important conclusion is that the ability to obtain feasible solutions does not imply the ability to minimise the makespan. We can notice several things:

1 The ranking of the algorithms is essentially the same in all columns.

2 The worst algorithm is Random+P though it is in a middle position in the ranking in relation to the first measure.

3 The best algorithm in the first three columns is Hartmann. However, where more instances are included into the comparison, presumably more

Table 4.10. Algorithm comparison with and without double justification.

	sols.	sols. + DJ	diffe-rence	sols in both	avr_dev	avr_dev
Random+S	55 (8)	81 (8)	26	48	9.29	5.99
Random+P	210 (4)	186 (4)	-24	172	7.66	5.90
B+LFT+S	44 (9)	57 (9)	13	33	8.54	4.24
B+LFT+P	148 (6)	153 (6)	5	127	6.25	4.92
B+WCS+P	143 (7)	148 (7)	5	128	6.32	5.34
B+EDD+P	281 (2)	272 (2)	-9	256	6.81	5.56
B+WCS_dd+P	280 (3)	264 (3)	-16	253	6.84	5.57
Hartmann	187 (5)	183 (5)	-4	147	5.34	2.30
Hartmann (2)	**322 (1)**	**327 (1)**	5	205	5.74	3.19

difficult ones, Hartmann(2) is the best. Considering both measures of quality, it seems that Hartmann(2) is the best algorithm.

4.4.2 The justification

The DJ (by extremes) has been successful in most algorithms for the TardinessRCPSP. However, the set where it has been less successful has been the loose set, precisely the one used in the computational study for the DeadlineRCPSP. Nevertheless, the DJ is very useful in the RCPSP when minimising the makespan. So, it is not clear in advance how its application to the DeadlineRCPSP is going to work. Table 4.10 compares the algorithms with and without DJ. For any algorithm given in column 1, the number of instances where this algorithm (this algorithm with DJ) obtains a feasible solution is given in column 2 (3). The ranking is shown in parenthesis. The fourth column displays the difference between columns 2 and 3. Column 5 shows the instances where both algorithms obtain a feasible solution. Column 6 (7) contains the average deviation in percentage between the makespan calculated by the algorithm (algorithm + DJ) with respect to the best solution known. Only instances where both algorithms obtain a feasible solution have been considered.

The DJ improves 5 algorithms and worsens 4 with respect to the first measure of quality. If we consider the second measure, all algorithms are improved if the DJ is applied. So, globally speaking, the DJ is useful in the DeadlineRCPSP. A very important fact is that DJ improves the best heuristic algorithm, Hartmann (2) in both measures. So, DJ is also able to improve good (metaheuristic) algorithms.

In Table 4.11 we compare with respect to the second measure all algorithms considered in Table 4.10 but the two worst according to this table. The meaning of the entries in Table 4.11 is the same as in Table 10.9.

Table 4.11. Average deviation in percentage with respect to the best solution known.

	114	119	140	179	245	258
B+WCS+P+DJ	5.47 (6)					
B+LFT+P+DJ	5.10 (4)	5.18 (5)				
Hartmann+DJ	**2.14 (1)**	**2.13 (1)**	**2.20 (1)**			
Random+P+DJ	5.70 (7)	5.76 (6)	6.01 (5)	5.94 (4)		
B+WCS_dd+P+DJ	5.00 (3)	5.02 (3)	5.35 (3)	5.22 (3)	5.50 (3)	
B+EDD+P+DJ	5.17 (5)	5.17 (4)	5.42 (4)	5.21 (2)	5.42 (2)	5.48 (2)
Hartmann(2)+DJ	2.63 (2)	2.67 (2)	2.80 (2)	**3.04 (1)**	**3.42 (1)**	**3.42 (1)**

Table 4.12. Number of feasible solutions and ranking.

combinations of rules	R1-L1	R1-L6	R3-L3	R2-L3	R4-L2	R4-L6
Hartmann + DJ	183 (12)	282 (9)	231 (11)	277 (10)	304 (8)	336 (5)
Hartmann(2) + DJ	327 (7)	342 (3)	341 (4)	344 (2)	333 (6)	**348 (1)**

The analysis of Table 4.11 leads to the same conclusions as those obtained from Table 10.9.

4.4.3 Justification by eligibles

We have applied the DJ by eligibles to the Hartmann and Hartmann(2) algorithms. We have chosen 6 of the best combinations of rules for the loose set according to Table 10.7 including the combination R1-L1 (DJ by extremes). Table 4.12 contains the results obtained concerning the first measure and the ranking.

We can observe that Hartmann(2) + DJ outperforms Hartmann + DJ for all combinations of rules; the justification by eligibles outperforms the justification by extremes, and the best algorithm is Hartmann(2)+DJ R4-L6 as far as the first measure of quality is concerned.

Table 4.13 takes into account the second measure, for the best 10 algorithms of the 12 considered in Table 4.12.

The best heuristic algorithms for the second measure are Hartmann+DJ R1-L6, Hartmann+DJ R4-L6, Hartmann(2)+DJ R1-L6 and Hartmann(2)+DJ R4-L6 if we do not take into account the fact that different instance sets are considered when evaluating the algorithms. The rest of the algorithms are not attractive, because Hartmann(2) + DJ R4-L6 improves them in both measures.

We have compared these 4 algorithms in twos. Table 4.14 shows the results. Numerical entry i, j shows the percentage average deviation of the makespan of the algorithm in column j with respect to that of the algorithm in row i. The average is computed over the set of instances for which algorithms obtain feasible solutions.

Table 4.13. Average percentage deviation from the best known solution.

	213	230	258	272	282	294	305	315	327
H+*DJ* R2-L3	2.76(5)								
H+*DJ* R1-L6	**0.90(1)**	**0.93(1)**							
H+*DJ* R4-L2	3.41(9)	3.36(6)	3.20(5)						
H(2)+*DJ* R1-L1	3.19(7)	3.41(8)	3.45(7)	3.44(6)					
H(2)+*DJ* R4-L2	3.83(10)	3.91(9)	3.78(8)	3.86(7)	3.87(6)				
H+*DJ* R4-L6	1.75(2)	1.87(2)	**1.88(1)**	**1.98(1)**	**1.98(1)**	**2.02(1)**			
H(2)*DJ* R3-L3	2.86(6)	3.04(5)	3.11(4)	3.05(4)	3.05(4)	2.99(4)	2.99(3)		
H(2)+*DJ* R1-L6	1.86(3)	2.03(3)	2.08(2)	2.10(2)	2.10(2)	2.08(2)	**2.08(1)**	**2.06(1)**	
H(2)+*DJ* R2-L3	3.23(8)	3.37(7)	3.38(6)	3.35(5)	3.39(5)	3.39(5)	3.36(4)	3.34(3)	3.30(2)
H(2)+*DJ* R4-L6	2.11(4)	2.30(4)	2.27(3)	2.37(3)	2.42(3)	2.47(3)	2.50(2)	2.52(2)	**2.56(1)**

Table 4.14. Algorithm comparison.

	H+DJ R1-L6	H+DJ R4-L6	H(2)+DJ R1-L6	H(2)+DJ R4-L6
H+DJ R1-L6		1.12%	1.05%	1.56%
H+DJ R4-L6	-1.03%		0.03%	0.46%
H(2)+DJ R1-L6	-0.98%	0.04%		0.51%
H(2)+DJ R4-L6	-1.44%	-0.42%	-0.43%	

Table 4.15. Justification by eligibles versus justification by extremes.

	sols $+DJ$	sols.$+DJ$ eligibles	sols. in both	avr_dev DJ	avr_dev eligibles
Hartmann	183	282	176	2.18	0.90
Hartmann(2)	327	342	309	3.27	2.06

It is clear that algorithm Hartmann+DJ R1-L6 outperforms the other algorithms and algorithm Hartmann(2)+DJ 4-6 is worse than the other three again, if any consideration to the first quality measure is disregarded.

In Table 4.15, we compare the algorithms Hartmann and Hartmann(2) with justification by extremes and with justification by eligibles with the combination R1-L6. The first column shows the considered algorithm. The second and third columns show the number of instances for which the algorithms with justification by extremes and by eligibles obtain feasible solutions respectively. The number of instances where the two types of justification reach a feasible solution is displayed in the fourth column. The two last columns show the average deviation with respect to the best solution known. We can observe that the algorithms with justification by eligibles outperform the algorithms with justification by extremes with respect to both measures of quality.

4.4.4 Comparison with a state-of-the-art GA

In section 2.8 of Neumann et al (2003), several procedures for the RCPSP with minimum and maximum time lag temporal constraints are compared. Taking into account all the instances considered, the best algorithm in quality is a genetic algorithm (GA_Max), followed by a tabu search. As already commented upon in the introduction, due dates can be modelled as temporal constraints with maximum time lags. In order to compare our results with GA_Max, we have programmed it and run it in the loose set. Our codification reaches different results than those obtained by the authors. In their computational experiment 4 instance sets with 90 instances each are considered, with 10, 20, 50 and 100

Table 4.16. Number of feasible solutions.

Algorithm	GA_Max	H+DJ 1-6	H+DJ 4-6	H(2)+DJ 1-6	H(2)+DJ 4-6
# feasible solutions	153	282	336	342	348

Table 4.17. Algorithms comparison.

	GA_Max	H+DJ 1-6	H+DJ 4-6	H(2)+DJ 1-6	H(2)+DJ 4-6
GA_Max		-4.04%	-3.73%	-3.11%	-3.22%
H+DJ 1-6	4.36%				
H+DJ 4-6	3.99%				
H(2)+DJ 1-6	3.32%				
H(2)+DJ 4-6	3.43%				

activities respectively. An average deviation with respect to the lower bound of 6.9% is obtained, whereas we reach 7.2%.

GA_Max is, up to a certain point, a generalisation of the GA of Hartmann. Instead of using one population, GA_Max works on several populations or islands (see, e. g., Dorigo and Maniezzo (1993), each of them created by sampling with a different priority rule. Each subpopulation evolves separately, until, after a given number *mig* of iterations, an individual migrates from one island to another. Another difference is the use of the one as well as the two-point crossover – an adaptation of those in order not to break the cycles that arise from having minimum and maximum time lags.

GA_Max performs a pre-processing phase before the proper evolution algorithm starts and then calculates at most 5000 schedules. It has two other ways of stopping the search, if several iterations have passed without improving the best solution, or if no feasible solution has been found in the first iterations. We have skipped these stopping criteria, so that the algorithm needs more time on average, but it is capable to obtain a feasible solution in more instances and better solutions. Since it generates a maximum of 5000 schedules, we can compare this algorithm with ours, although the pre-processing phase may need a significant amount of time.

The following tables compare GA_Max with our best algorithms.

We can see that any of the four algorithms with justification by eligibles outperforms GA_Max with respect to both quality measures.

4.5 Conclusions

In this paper we have studied two extensions of the RCPSP when due dates and deadlines are considered. The paper centres on the application of well-known RCPSP heuristics (and their adaptations) to both problems. Specifically, we have tested one or more representatives of three of the most important RCPSP heuristic types: priority rules, sampling procedures and metaheuristics. Also, we have analysed the effect of the double justification, nowadays used in most heuristics for the RCPSP. The double justification can worsen solutions for the TardinessRCPSP and make unfeasible a given feasible solution for the DeadlineRCPSP. Nevertheless, we have been able to outperform previous results through its incorporation. To further increase the improvements obtained with the application of double justification we have also applied a more general type of justification - justification by eligibles - recently proposed by the authors of this paper (Valls et al (2006)). The computational results show that the justification by eligibles outperforms the type of justification applied up to now - at least in these two newly studied problems. With this type of justification we have been able to outperform the results of state-of-the-art algorithms that can be applied to the DeadlineRCPSP.

We have also proposed an instance generator that assigns due dates to RCPSP instances, in particular to those in the standard set j120. It is able to generate three types of instance sets: (1) the loose set, with instances where it is easy to fulfil most of the due dates, (2) the tight set, with instances where many activities are going to be tardy in any schedule, and (3) the medium set, with instances in between. The different performance of the tested algorithms on the three sets supports the partition we have made.

Acknowledgement

This research was partially supported by the Ministerio de Ciencia y Tecnología under contract TIC2002-02510 and the Agencia Valenciana de Ciencia y Tecnología, GRUPOS03/174.

References

Ballestín, F. (2002). Nuevos métodos de resolución del problema de secuenciación de proyectos con recursos limitados, Unpublished PhD Dissertation, Universidad de Valencia.

Blazewicz, J., Lenstra, J.K., and Rinooy Kan, A.H.G. (1983). Scheduling subject to resource constraints: classification and complexity, *Discrete Applied Mathematics*, 5:11–24.

Brucker, P., Drexl, A., Möhring, R., Neumann K. and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research*, 112:3–41.

Dorigo, M., and Maniezzo, V. (1993). Parallel Genetic Algorithms: Introduction and Overview of Current Research, in: *Parallel Genetic Algorithms: Theory & Applications*, J. Stenders, ed., IOS Press, Amsterdam, pp. 5–42.

Garey, M. R., and Johnson, D. S. (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company. San Francisco.

Gordon, V., Proth, J-M., and Chu, C. (2002). A survey of the state-of-the art of common due date assignment and scheduling research, *European Journal of Operational Research*, 139:1–25.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling, *Naval Research Logistics*, 45:733–750.

Herroelen, W., Demeulemeester, E., and De Reyck, B. (1998). A Classification scheme for project scheduling, in: *Project Scheduling. Recent Models, Algorithms and Applications*, J. Weglarz, ed., Kluwer Academic Publishers, pp. 1–26.

Jackson, J. R. (1955). Scheduling a production line to minimize maximum tardiness, Management Science Research Project, Technical Report No. 43, UCLA.

Kolisch, R. (1995). *Project Scheduling under Resource Constraints - Efficient Heuristics for several Problem Classes*, Phisica, Heidelberg.

Kolisch, R. (1996). Efficient priority rules for the resource-constrained project scheduling problem, *Journal of Operations Management* 14:179–192.

Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems, *Management Science*, 41:1693–1703.

Kolisch, R., and Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis, in: *Project Scheduling. Recent Models, Algorithms and Applications*, Weglarz, J., ed., Kluwer Academic Publishers, Boston, pp. 147–178.

Kolisch, R., and Sprecher, A. (1997). PSPLIB - A project scheduling library, *European Journal of Operational Research*, 96:205–216.

Koulamas, C. (1994). The total tardiness problem: Review and extensions, *Operations Research*, 42:1025–1041.

Li, KY., and Willis, RJ. (1992). An iterative scheduling technique for resource-constrained project scheduling, *European Journal of Operational Research*, 56:370–379.

Neumann, K., Schwindt, C., and Zimmermann, J. (2003). *Project Scheduling with Time Windows and Scarce Resources*, 2nd edition, Springer Verlag, Berlin.

Özdamar, L., and Ulusoy, G. (1996). An iterative local constraint based analysis for solving the resource constrained project scheduling problem, *Journal of Operations Management*, 14:193–208.

Schirmer, A., and Riesenberg, S. (1997). Parameterized Heuristics for Project Scheduling - Biased Random Sampling Methods, Technical report 456, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel.

Thomas, P.R., and Salhi, S. (1997). An investigation into the relationship of heuristic performance with network-resource characteristics, *Journal of the Operational Research Society*, 48:34–43.

Tormos, P., and Lova, A. (2001). A competitive heuristic solution technique for resource-constrained project scheduling, *Annals of Operations Research*, 102:65–81.

Valls, V., Ballestín, F., and Quintanilla, S. (2005a), Justification and RCPSP: a technique that pays, *European Journal of Operational Research*, 165:375–386.

Valls, V., Ballestín, F., and Quintanilla, S. (2006). Justification Technique Generalisations, in: *Perspectives in Modern Project Scheduling*, Józefowska J., and Weglarz J., eds, Kluwer, pp. 205–223.

Vanhoucke, M. (2002). Optimal due date assignment in project scheduling, Vlerick Leuven Gent Management School Working Paper Series 2002-19, Vlerick Leuven Gent Management School.

Vanhoucke, M., Demeulemeester, E., and Herroelen, W. (1999). Exact procedure for the unconstrained weighted earliness-tardiness project scheduling problem, Research Report. Department of Applied Economics, Katholieke Universiteit , Leuven, Belgium.

Vanhoucke, M., Demeulemeester, E., and Herroelen, W. (2001). Exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem, *Annals of Operations Research*, 102:179–196.

Chapter 5

RCPS WITH VARIABLE INTENSITY ACTIVITIES AND FEEDING PRECEDENCE CONSTRAINTS*

Tamás Kis

Computer and Automation Institute, Hungarian Academy of Sciences
1111 Budapest, Kende str. 13-17, Hungary
tamas.kis@sztaki.hu

Abstract This paper presents a branch-and-cut based exact solution algorithm for scheduling of projects with variable intensity activities connected by feeding precedence constraints the objective being to minimize the violation of resource constraints. Feeding precedence constraints allow some overlap in the execution of the connected activities and capture the flow of material or information between them. New polyhedral results are obtained and computational results are summarized.

Keywords: variable intensity activities, branch-and-cut.

5.1 Introduction

Project scheduling with variable intensity activities aims at the allocation of continuously divisible resources to activities over time in varying quantities such that a set of temporal and resource constraints are satisfied and an objective function is minimized. In this model there is a finite set of activities, N, to be performed using a finite set of continuously divisible resources, R. The time horizon is divided into discrete time periods $[t, t+1)$, $t \in \{1, \ldots, T\}$. The intensity of each activity $i \in N$ may vary over time: it is bounded from below by 0, and from above by a constant $a^i \leq 1$. If x_t^i denotes the chosen intensity of activity i in time period $[t, t+1)$, then $0 \leq x_t^i \leq a^i$ has to hold. Each activity i has a release time r^i and a deadline d^i. Namely, r^i and d^i refer to the first and last time periods, respectively, in which activity i may be

*This research was supported by the János Bolyai research grant BO/00380/05 and by OTKA T046509.

processed, that is, $x_t^i = 0$ for all $t < r^i$ and for all $t > d^i$, and $\sum_{t=r^i}^{d^i} x_t^i = 1$. Notice that $r^i \geq 1$ and $d^i \leq T$ for all $i \in N$. In the basic model there are finish-to-start with zero time lags precedence constraints between pairs of activities meaning that the predecessor activity has to be completed before the successor activity may be started. Each activity requires a constant mix of some renewable and/or non-renewable resources during its execution and the demand from every resource is proportional to the intensity of the activity in every time period. That is, if the total demand of activity i from resource k is q_k^i, then it requires $q_k^i \cdot x_t^i$ units of resource k in time period $[t, t+1)$. For simplicity in this paper we consider only renewable resources, where the capacity of resource k in time period $[t, t+1)$ is constant b_t^k. It is permitted that the total demand from a resource k in a time period $[t, t+1)$ exceeds its capacity b_t^k, that is, $y_t^k = \max\{0, (\sum_{i \in N} q_k^i \cdot x_t^i) - b_t^k\}$ may be greater than zero. The cost of over-using resource k in time period t is c_t^k. The objective is to minimize the cost of violating the resource constraints, $\min \sum_{k \in R} \sum_t c_t^k \cdot y_t^k$, subject to the intensity and the precedence constraints of the activities.

As already mentioned, in the basic model there are finish-to-start precedence constraints between pairs of activities. However, in applications it may be desirable to allow some overlap in the execution of two activities connected by a precedence constraint. For instance, activity j may be started only if, say, 30% of activity i has been completed. Then $x_\ell^j = 0$ has to hold for all ℓ such that $\sum_{t<\ell} x_t^i < 0.3$. In addition, to model the flow of material or information between i and j it is also required that the fraction of activity j done by time t is never greater than that of activity i by time t, for any time point t, i.e., $\sum_{t \leq \ell} x_t^j \leq \sum_{t \leq \ell} x_t^i$, for any integer $\ell \geq 0$. We will also say that activity i *feeds* activity j. In general, a *feeding precedence constraint* is given by a triple (i, j, f_{ij}), where i and j are activities and $0 \leq f_{ij} \leq 1$. Such a constraint specifies that $x_\ell^j = 0$ has to hold for all ℓ with $\sum_{t<\ell} x_t^i < f_{ij}$, and i feeds j. The special case with $f_{ij} = 1$ corresponds to the well-known finish-to-start precedence constraint. The other extremity, $f_{ij} = 0$, means that activities i and j may be started simultaneously, and i feeds j.

The partial overlap between successive activities may, in principle, be modeled by splitting the predecessor activity into two parts, and then introducing appropriate finish-to-start precedence constraints. For instance, to model the feeding precedence constraint $(i, j, 0.3)$, one could split activity i into two parts, one representing 30%, another representing 70% of activity i. Denoting by i_1 and i_2 these two parts, we add finish-to-start precedence constraints between i_1 and i_2, and between i_1 and j. Such a model would have two major disadvantages. On the one hand, the number of activities would grow considerably, increasing the computational complexity of the problem. On the other hand, since the time horizon is divided into discrete time periods, it is not possible in general to divide an activity into two activities representing exactly, say, 30% and 70% percent of the original activity. To avoid these anomalies, we model feeding precedence constraints in a different way by generalizing some of the results in Kis (2004). We will model the problem by a mixed-integer linear program and solve it by cutting plane techniques.

Typical applications of this model are project planning in a shipyard (Leachman et al (1990)), or production planning in a make-to-order manufacturing environment, where each customer order is a project and the company resources are aggregated at a high level (Márkus et al (2003)). With feeding precedence constraints we can model situations where the output of the predecessor activity is fed into the successor activity after some initial quantity is produced.

In the following we describe a new problem formulation (Section 5.2), summarize previous work (Section 5.3), introduce new cutting planes (Section 5.4), and provide computational results (Section 5.5). We draw some conclusions in Section 5.6.

5.2 Problem formulation by mixed-integer program

We assume that the time horizon is divided into T periods, $\min r^i = 1$ and $\max d^i = T$. For any $i \in N$, let $A(i)$ be the set of those precedence constraints (k, j, f_{kj}) with $k = i$. Clearly, $A(i)$ may contain distinct constraints (i, j, f_{ij}) and (i, k, f_{ik}) with $j \neq k$ and $f_{ij} = f_{ik}$. Let $F^i = \{f \in [0, 1] \mid \exists (i, j, f_{ij}) \in A(i) \text{ with } f = f_{ij}\}$ be the set of distinct f_{ij} values associated with the precedence constraints in $A(i)$. Let p^{if} denote the smallest integer such that $p^{if} \cdot a^i \geq f$. Notice that $p^{if} \geq 1$ when $f > 0$, otherwise, when $f = 0$, $p^{if} = 0$. In order to model feeding precedence constraints, for each activity i, and $f \in F^i$ we introduce bunches of binary decision variables z_t^{if}, $t = r^i + p^{if}, \ldots, d^i$.

$$z_t^{if} = \begin{cases} 1 & \text{if less than an } f \text{ fraction of activity } i \text{ is processed up to time} \\ & t. \\ 0 & \text{otherwise.} \end{cases}$$

Since the earliest time point when at least an f fraction of activity i is processed is $r^i + p^{if}$, we do not include the variables z_t^{if} with $t \in \{r^i, \ldots, r^i + p^{if} - 1\}$ into our model.

In addition, the decision variables x_t^i represent the intensity of activity i in time periods $[t, t+1)$, $t \in \{r^i, \ldots, d^i - 1\}$, whereas the variables y_t^k measure the violation of the capacity constraints associated with resource k in the periods $t \in \{1, \ldots, T\}$. The mixed-integer program (MIP) for our problem is the following:

$$\min \sum_{k \in R} \sum_{t=1}^{T} c_t^k y_t^k \tag{5.1a}$$

subject to

$$\sum_{t=r^i}^{d^i} x_t^i = 1, \quad i \in N, \tag{5.1b}$$

$$\sum_{t=r^i}^{\ell-1} x_t^i \geq f(1 - z_\ell^{if}), \quad i \in N,\ f \in F^i,\ \ell \in \{r^i + p^{if}, \ldots, d^i\}, \tag{5.1c}$$

$$x_t^j \leq a^j(1 - z_t^{if}), \quad i \in N,\ (i,j,f) \in A(i), \tag{5.1d}$$

$$\sum_{t=r^i}^{\ell} x_t^i \geq \sum_{t=r^j}^{\ell} x_t^j, \quad (i,j,f) \in A(i),\ \ell \in \{\max\{r^i, r^j\}, \ldots, \min\{d^i, d^j\}\}, \tag{5.1e}$$

$$z_t^{if} \geq z_{t+1}^{if}, \quad i \in N,\ f \in F^i,\ t \in \{r^i + p^{if}, \ldots, d^i - 1\}, \tag{5.1f}$$

$$\sum_{i \in N_t^k} q_k^i \cdot x_t^i \leq b_t^k + y_t^k, \quad k \in R, t \in \{1, \ldots, T\}, \tag{5.1g}$$

$$0 \leq x_t^i \leq a^i, \quad i \in N,\ t \in \{r^i, \ldots, d^i\}, \tag{5.1h}$$

$$0 \leq y_t^k \leq \bar{b}_t^k, \quad k \in R,\ t \in \{1, \ldots, T\}, \tag{5.1i}$$

$$z_t^{if} \in \{0,1\}, \quad i \in N,\ f \in F^i,\ t \in \{r^i + p^{if}, \ldots, d^i\}, \tag{5.1j}$$

where $N_t^k = \{i \in N \mid q_k^i > 0,\ r^i \leq t \leq d^i\}$ is the set of activities that may require resource k in time period t. The objective (5.1a) is to minimize the penalty cost of violating the resource capacity constraints. Equations (5.1b) express that all activities must entirely be processed between their release date and deadline. The precedence constraints are expressed by constraints (5.1c) through (5.1f). Namely, consider any $(i,j,f) \in A(i)$. (5.1c) ensures that $z_\ell^{if} = 1$ for all ℓ with $\sum_{t=r^i}^{\ell-1} x_t^i < f$. On the other hand, (5.1d) guarantees that $x_\ell^j = 0$ for all ℓ with $z_\ell^{if} = 1$, while (5.1e) yields that i feeds j. Finally, by (5.1f) there is a time point ℓ such that $z_t^{if} = 1$ for all $t < \ell$ and $z_t^{if} = 0$ for all $t \geq \ell$. These constraints also ensure that the differences $z_t^{if} - z_{t+1}^{if}$, $t \in \{r^i + p^{if}, \ldots, d^i - 1\}$ are all non-negative, which will be important for our separation algorithms. The violations of resource capacity constraints are measured in (5.1g). The domains of the variables are given in the last three constraints. The upper bounds $\bar{b}_t^k$ on the variables y_t^k may take the value $+\infty$, in which case y_t^k is unbounded.

5.2.1 Preprocessing

Preprocessing is part of the computations done on a problem instance and is usually discussed in the algorithmic part of a paper. However, the material below influences our polyhedral investigations by determining certain parameters of the polytopes studied. Therefore, we do preprocessing prior to theory.

First, when $0 \in F^i$, there is no need for the variables $z_t^{i,0}$, and these variables along with all constraints involving any of them can be eliminated. A further reduction is possible by using the earliest and latest start times of the activities, respectively. These parameters can be computed, as usual, by first determining a topological order of the activities with respect to the precedence constraints and then scanning the list forward and backward, respectively. In the forward pass the *earliest start time*, $est(j)$, of activity j is determined by the recursive formula:

$$est(j) = \max\{r^j, \max\{est(i) + p^{if} \mid \exists i \in N,\ (i, j, f) \in A(i)\}\}.$$

Let p^i be the smallest integer with $p^i \cdot a^i \geq 1$. The *latest start time*, $lst(i)$, of activity i is computed in the backward pass as follows:

$$lst(i) = \min\{d^i - p^i + 1, \min\{lst(j) - p^{if} \mid \exists i \in N,\ (i, j, f) \in A(i)\}\}.$$

Using the values computed above, the release times and the deadlines of the activities can be tightened. Namely, for any activity i, r^i can be increased to $est(i)$. Moreover, d^i can be decreased to $lst(i) + p^i - 1$, provided that there exists $(i, j, f) \in A(i)$ with $f = 1.0$, in which case $p^{if} = p^i$ and the activity must complete by $lst(i) + p^i$. In addition, the variables z_t^{if} need be defined only for $t \in \{r^{if}, \ldots, d^{if}\}$, where $r^{if} = est(i) + p^{if}$ and $d^{if} = lst(i) + p^{if} - 1$. If $f < 1.0$, we add the constraint $\sum_{t=r^i}^{d^{if}} x_t^i \geq f$ to the model.

Since $est(i)$ and $lst(i)$ can be computed in linear time, the time complexity of the reduction is linear in the size of the problem.

A necessary condition for the existence of a feasible solution is that $est(i) \leq lst(i)$ for all $i \in N$. This condition is also sufficient when all y_t^k are unbounded. Therefore, preprocessing may provide important information about the feasibility status of the problem at hand.

We close this section with one more check for eliminating some variables and constraints. Namely, for any resource k and time point t, the variable y_t^k can be set to 0, and therefore it can be eliminated from MIP, whenever $\sum_{i \in N_t^k} a^i q_k^i \leq b_t^k$ holds.

5.3 Previous work

The first papers on project scheduling with variable intensity activities studied models with complex relationships between the intensities of the activities and the amount of resources allocated to them over time. Weglarz defined a very general model consisting of n activities characterized by equations

$$\frac{dx_i(t)}{dt} = \begin{cases} f_i[r_i(t)] & \text{if } t \in [t_i, T_i], \\ 0 & \text{otherwise,} \end{cases} \qquad i = 1, \ldots, n,$$

where $x_i(t)$ and $r_i(t)$ are the state and resource usage of activity i at time point t, respectively, $x_i(t_i) = 0$, $f_i(\cdot)$ is a continuous, increasing function, $f_i(0) = 0$, and t_i, T_i are the starting and finishing times of activity i, unknown in advance (see Weglarz (1976) and Weglarz (1979)). The final state of activity i is x_i, that is,

$$\int_0^T f_i[r_i(t)]dt = \int_{t_i}^{T_i} f_i[r_i(t)]dt = x_i, \quad i = 1, \ldots, n.$$

Here, T is the project duration.

The value of x_i is an objective measure of work needed to complete activity i, such as total number of working hours to perform i. As can be seen, the process is driven by the allocation of resources to activities, $r_i(t)$, $i = 1, \ldots, n$, over time, that we have to determine. The resource usage is related to the progress of activity i through the function $f_i(\cdot)$. In the simplest case there is only one resource, which is assumed to be doubly constrained:

$$\sum_{i=1}^{n} r_i(t) \leq B \qquad \text{for every } t \geq 0,$$

and

$$\sum_{i=1}^{n} \int_0^T r_i(t)dt \leq M,$$

where B, M are known constants, bounding the resource usage in any time point t, and the total usage over the entire project duration, respectively.

The solution to the above problem is a vector function $r(t) = (r_1(t), \ldots, r_n(t))$, $r_i(t) \geq 0$, which fulfills the above two types of resource constraints and minimizes the project duration T. Weglarz determined necessary and sufficient conditions as well as the optimal $r^*(\cdot)$ under the assumption that all the $f_i(\cdot)$ are convex or all the $f_i(\cdot)$ are concave. Furthermore, using an activity-on-arc network, Weglarz extended his results to dependent activities. However, the method is impractical for larger projects.

When f_i is linear, x_i can be expressed in [resource $\times$ time] units such as man-days. Models of this type has been studied in Leachman et al (1990) (see also Leachman (1983)). It is assumed that resources are of finite capacity, non-renewable and arbitrarily divisible among activities. Moreover, in each moment of time the mix of applying the different types of resources required by an activity is exactly proportional to the mix of total resource requirements to complete the activity. Therefore, the *intensity* of an activity at time t is the rate of utilization of the required resources. The intensity of any activity i can be 0, i.e., the activity is not started or already finished, or it can take any value from a continuous interval $[\underline{a}^i, \overline{a}^i]$, where $0 \leqslant \underline{a}^i \leqslant \overline{a}^i \leqslant 1$. Notice that unlike in the model studied in the present paper, time is continuous in the model studied by Leachman et al The authors proposed a single-phase and a two-phase algorithm for minimizing the project duration. Both algorithms build a resource-feasible schedule forward in time by adjusting the intensities of the activities at decision points. A computational evaluation revealed the superiority of the latter heuristic. The methods were tested on problem instances up to 3000 activities served by 50 resources, but without any comparison to some lower bound on project duration.

Leachman et al also discussed a model with overlapping execution of activities, that they call "production-like workflow dependencies". In that model, for certain pairs of activities i and j there is given a progress lag f_{ij}, $0 < f_{ij} \leqslant 1$, which is to be interpreted as follows: while less than an f_{ij} fraction of activity i is completed, activity j cannot be started. Then until activity i is completed, the fraction of activity j completed by time t is at most the fraction of activity i done by time t minus f_{ij}. Once activity i is completed, there is no additional restriction on the progress of activity j.

In the remainder of this section we review several approaches for solving the same model with the following characteristics: (1) the time horizon is divided into discrete time periods, $1, \dots, T$, each of the same length, (2) the intensity of each activity i may vary continuously between 0 and a^i, where $a^i \leq 1$ is a constant, and (3) there are finish-to-start precedence constraints between the activities.

Tavares modeled the precedence constraint $(i, j) \in A$ by a set of constraints equivalent to the following (Tavares (1998), Tavares (2002)):

$$\left(\sum_{t=t_0}^{d^i} x_t^i \right) x_{t_0}^j = 0, \quad \forall\, t_0 \in \{r^j, \dots, d^j\}.$$

Tavares showed that the project duration can be minimized by defining new auxiliary variables z_t with $0 \leq z_t \leq 1$, $t \in \{1, \dots, T\}$, and adding the constraints

$$\left(\sum_{i \in N} \sum_{t=t_0}^{T} x_t^i \right) z_{t_0} = 0, \quad \forall\, t_0 \in \{1, \dots, T\}.$$

to the model. Notice that if $z_{t_0} > 0$, then no activity can be performed after t_0. Therefore, the above constraints along with the objective $\min(T - \sum_t z_t)$ express the makespan minimization problem. Since the constraints are clearly non-linear in the variables x and z, Tavares used a non-linear optimization software for solving the problem w.r.t. the makespan objective.

Branch-and-price based exact solution approaches are described in Hans (2001) for several variants of the problem with finish-to-start precedence constraints and a slight relaxation thereof. Here we focus on the *rough cut capacity planning* problem, which is the same model as ours, except that there can be no overlap between activities connected by a precedence constraint. For this problem, Hans devised an efficient pricing algorithm to be used in his branch-and-price algorithm. He obtained an initial upper bound by various constructive heuristics and by local search. In his approach, the precedence constraints are modeled by a set of binary vectors $\{\beta^h \in \{0,1\}^{|N|\times T} \mid h \in \Pi\}$, Π being a suitable set of indices, consisting of the supports of all feasible intensity assignments to the activities. Albeit Hans considered distinct projects among which there were no precedence constraints, to simplify notation we assume that all activities belong to the same project. Notice that a binary vector $\beta \in \{0,1\}^{|N|\times T}$ is the support of a feasible intensity assignment if and only if $\sum_{t=1}^{T} \beta_{i,t} \geq p^i$, $\min\{t \mid \beta_{i,t} = 1\} \geq r^i$, $\max\{t \mid \beta_{i,t} = 1\} \leq d^i$, and if $(i,j) \in A$, then $\max\{t \mid \beta_{i,t} = 1\} < \min\{t \mid \beta_{j,t} = 1\}$. For solving the problem, precisely one vector β^h must be chosen. To this end, Hans introduced new binary variables z_h, $h \in \Pi$, together with the following constraints:

$$\begin{aligned}
\sum_{h\in\Pi} z_h &= 1, \\
z_h &\in \{0,1\}, \quad h \in \Pi, \\
0 \leq x_t^i &\leq a^i \left(\sum_{h\in\Pi} \beta_{i,t}^h z_h \right), \quad i \in N,\ t \in \{r^i, \ldots, T\}.
\end{aligned}$$

The first two constraints ensure that exactly one vector β^h is chosen. The third one specifies that x_t^i is either 0, or is between 0 and a^i, depending on whether $\beta_{i,t}^h$ is 0 or 1. Hans' model incorporates resource constraints similar to ours, although instead of (5.1i) it has $y_t^k \geq 0$, and $\sum_k y_t^k \leq \bar{b}^k$, for all t. As the size of Π can be enormous, column generation is the only viable approach to handle this formulation.

The primary advantage is that any objective function which depends linearly on the cost associated with the vector β^h, whatever this cost be, can be optimized, provided that the pricing problem can be solved efficiently. The main drawback is that a huge number of columns must be handled, and enlarging all activity deadlines by only one time period may multiply the size of Π which may increase considerably the running time of the pricing algorithm.

Linear programming based heuristics are proposed in Gademann and Schutten (2005). It is assumed that there are no limitations on the violation of resource capacities ($\bar{b}_t^k = \infty$). The authors classified the heuristics into three categories: (1) constructive, (2) repair, and (3) iterative improvement using shadow prices from LP solution. A central notion is the *allowed to work* time window (ATW), which is merely an interval of time periods in which an activity may be processed. A collection of ATWs (one for each activity) is feasible if it respects the release times and deadlines of the activities, and the ATWs of activities connected by a precedence constraint are disjoint and are in the right order. The constructive heuristics first determine a feasible ATW for each activity, then solve the linear program in which activities can only be processed in their respective ATWs. The repair heuristic algorithm starts with a collection of ATWs which are not feasible, but give the best objective function value. Then it fixes the infeasibilities by processing one-by-one those pairs of activities connected by a precedence constraint and with overlapping ATWs. For each such pair, it selects the best way (giving the smallest objective function value) to separate the ATWs of the two activities by enumerating all possible time points for separation. Finally, the iterative heuristic method first determines a feasible collection of ATWs, then it tries to change the windows in a local search algorithm. By combining the second and third approach, Gademann and Schutten obtained the currently best heuristic approach, which first seeks a feasible collection of ATWs by repair, then performs a limited local search by exploring the neighborhood of the solution returned by repair.

In Chapter 3 of Wullink (2005), a new constructive heuristics is proposed for the problem that seems to outperform all other constructive heuristics. The advantage of his approach to other methods is its speed as it obtains good results much faster than e.g., the powerful heuristic approach of Gademann and Schutten.

In Kis (2004), the author obtained polyhedral results for non-over-lapping precedence constraints. In the following overview we adapt the notation to the present paper and discuss only those results that are enhanced in the next section. A complete description of the polytope K^{ij} of feasible intensity assignments to a pair of activities $(i, j) \in A$ is established along with fast separation algorithms. That is, K^{ij} is the convex hull of all points $(x^i, x^j, z^i) \in \mathbb{R}^{s^i} \times \mathbb{R}^{s^j} \times \{0,1\}^{s^i - p^i}$, where $s^i = d^i - r^i + 1$ and $s^j = d^j - r^j + 1$, satisfying the following constraints:

$$\sum_{t=r^i}^{d^i} x_t^i = 1, \tag{5.2}$$

$$0 \le x_t^i \le a^i, \quad t \in \{r^i, \dots, r^i + p^i - 1\}, \tag{5.3}$$

$$0 \le x_t^i \le a^i \cdot z_t^i, \quad t \in \{r^i + p^i, \dots, d^i\}, \tag{5.4}$$

(5.5)

$$
\begin{aligned}
z_t^i &\geq z_{t+1}^i, \quad t \in \{r^i + p^i, \ldots, r^j - 1\}, && (5.6)\\
z_t^i &\geq z_{t+1}^i, \quad t \in \{r^j, \ldots, d^i - 1\}, && (5.7)\\
\sum_{t=r^j}^{d^j} x_t^j &= 1, && (5.8)\\
0 \leq x_t^j &\leq a^j \cdot (1 - z_t^i), \quad t \in \{r^j, \ldots, d^i\}, && (5.9)\\
0 \leq x_t^j &\leq a^j, \quad t \in \{d^i + 1, \ldots, d^j\}. && (5.10)
\end{aligned}
$$

To facilitate the analysis of K^{ij}, the polytope K is defined as the convex hull of those points $(x, z) \in \mathbb{R}^n \times \{0,1\}^{n-q}$, where $q < n$ satisfies $q \cdot a \geq 1$ ($q \cdot a$ may be much bigger than 1):

$$
\begin{aligned}
\sum_{t=1}^{n} x_t &= 1, \\
x_t &\leq a z_t, \quad t \in \{q+1, \ldots, n\}, && (5.11)\\
x_t &\leq a, \quad t \in \{1, \ldots, q\}, \\
z_t &\geq z_{t+1}, \quad t \in \{q+1, \ldots, n-1\}, \\
x_t &\geq 0, \quad t \in \{1 \ldots, n\}.
\end{aligned}
$$

Notice that z is indexed from $q+1$ to n. For stating the next result, define the vectors $z^\ell \in \{0,1\}^{n-q}$:

$$
z_t^\ell = \begin{cases} 1 & \text{if } t \in \{q+1, \ldots, \ell\}, \\ 0 & \text{if } t \in \{\ell+1, \ldots, n\}, \end{cases} \qquad \ell \in \{q, \ldots, n\}.
$$

LEMMA 5.1 *Let (x, z) be any point in $\mathbb{R}^n \times \mathbb{R}^{n-q}$. Then $(x, z) \in K$ if and only if the numbers $\lambda_q = 1 - z_{q+1}$, $\lambda_\ell = z_{\ell-1} - z_\ell$ ($\ell \in \{q+1, \ldots, n-1\}$) and $\lambda_n = z_n$ are all non-negative and there exist vectors $x^\ell \in \mathbb{R}^n$, $\ell \in \{q, \ldots, n\}$, such that $\sum_{\ell=q}^{n} \lambda_\ell x^\ell = x$ and every (x^ℓ, z^ℓ) belongs to K.*

It can be shown that in general a minimal linear representation of K consists of the defining inequalities and also the following set of inequalities:

$$
a_r z_{t_1} + a \sum_{t \in S_1 \setminus \{t_1\}} z_t + \sum_{t \in \{1, \ldots, t_1\} \setminus (S_1 \cup S_2)} x_t \geq 1 - a|S_2|, \qquad (5.12)
$$

where $a(p-1) < 1 \leq ap$, $a_r = 1 - (p-1)a$, $\emptyset \neq S_1 \subseteq \{q+1, \ldots, n\}$ and $S_2 \subseteq \{1, \ldots, q\}$ are such that $|S_1| + |S_2| = p$ and t_1 is the greatest element of S_1.

K^{ij} can be decomposed into two smaller dimensional polytopes, K^{i*} and K^{*j}:

$$\begin{aligned} K^{i*} &= \operatorname{conv}\left\{(x^i, z^i) \in \mathbb{R}^{s^i} \times \{0,1\}^{s^i - p^i} \mid (x^i, z^i) \text{ satisfies } (5.2) - (5.7)\right\}, \\ K^{*j} &= \operatorname{conv}\left\{(x^j, \tilde{z}^i) \in \mathbb{R}^{s^j} \times \{0,1\}^{d^i - r^j + 1} \mid (x^j, \tilde{z}^i) \text{ satisfies } (5.7) - (5.10)\right\}. \end{aligned}$$

It is easy to show that both of the above polytopes can be obtained from K by reindexing variables and by appropriate substitutions. K^{ij} has the following characterization:

LEMMA 5.2 *Let* (x^i, x^j, z^i) *be any point in* $\mathbb{R}^{s^i} \times \mathbb{R}^{s^j} \times \mathbb{R}^{s^i - p^i}$. *Then* $(x^i, x^j, z^i) \in K^{ij}$ *if and only if* $(x^i, z^i) \in K^{i*}$ *and* $(x^j, \tilde{z}^i) \in K^{*j}$, *where* $\tilde{z}^i_t = z^i_t$ *for all* $t \in \{r^j, \ldots, d^i\}$.

It can be shown that most of the facets of K^{i*} and K^{*j} will be facets of K^{ij}.

5.4 New polyhedral results

In Kis (2004), various polyhedral results are presented for the polytope of feasible intensity assignments to a pair of activities connected by a finish-to-start precedence constraint. Below we extend these results to feeding precedence constraints.

5.4.1 Feasible intensity assignments to a pair of activities connected by a feeding precedence constraint

Consider a precedence constraint $(i, j, f) \in A(i)$. In this section we study the polytope K^{ij}_f of feasible intensity assignments to i and j satisfying the precedence constraint (i, j, f). K^{ij}_f is the convex hull of those points (x^i, x^j, z^{if}) with x^i, x^j non-negative, z^{if} a 0/1 vector that satisfy the following system of inequalities:

$$\begin{aligned}
\sum_{t=r^i}^{d^i} x_t^i &= 1, & (5.13)\\
\sum_{t=r^i}^{\ell-1} x_t^i &\geq f \cdot (1 - z_t^{if}), \quad \ell \in \{r^{if}, \ldots, d^{if}\}, & (5.14)\\
\sum_{t=r^i}^{d^{if}} x_t^i &\geq f, & (5.15)\\
x_t^i &\leq a^i, \quad t \in \{r^i, \ldots, d^i\}, & (5.16)\\
z_t^{if} &\geq z_{t+1}^{if}, \quad t \in \{r^{if}, \ldots, r^j - 1\}, & (5.17)\\
z_t^{if} &\geq z_{t+1}^{if}, \quad t \in \{r^j, \ldots, d^{if} - 1\}, & (5.18)\\
\sum_{t=r^j}^{d^j} x_t^j &= 1, & (5.19)\\
x_t^j &\leq a^j(1 - z_t^{if}), \quad t \in \{r^j, \ldots, d^{if}\}, & (5.20)\\
x_t^j &\leq a^j, \quad t \in \{r^j, \ldots, d^j\}, & (5.21)\\
\sum_{t=r^i}^{\ell} x_t^i &\geq \sum_{t=r^j}^{\ell} x_t^j, \quad \ell \in \{\max\{r^i, r^j\}, \ldots, \min\{d^i, d^j\}\}. & (5.22)
\end{aligned}$$

In the definition of K_f^{ij}, z^{if} is restricted to 0/1 vectors. We will give a system of linear inequalities with set of solutions K_f^{ij} along with corresponding separation algorithms. These inequalities can be used to strengthen our MIP (after preprocessing) in a branch-and-cut algorithm. To this end, we define two lower dimensional polytopes. K_f^{i*} is the convex hull of vectors $(x^i, z^{if}) \in \mathbb{R}_+^{d^i - r^i + 1} \times \{0,1\}^{d^{if} - r^{if} + 1}$ that satisfy the constraints (5.13) through (5.18). In turn, K_f^{*j} is the convex hull of vectors $(x^j, \tilde{z}^{if}) \in \mathbb{R}_+^{d^j - r^j + 1} \times \{0,1\}^{d^{if} - r^j + 1}$ satisfying (5.18) through (5.21). In K_f^{i*} any vector (x^i, z^{if}) with integral z^{if} corresponds to a feasible intensity assignment in which at least an f fraction of activity i is completed by the end of period $[\ell, \ell + 1)$, where $\ell \in \{r^{if} - 1, \ldots, d^{if}\}$ is such that $z_t^{if} = 1$ for all $t \in \{r^{if}, \ldots, \ell\}$ (this interval is empty if $\ell = r^{if} - 1$), and $z_t^{if} = 0$ for all $t \in \{\ell + 1, \ldots, d^{if}\}$ (this interval is empty when $\ell = d^{if}$). In contrast, any vector $(x^j, \tilde{z}^{if}) \in K_f^{*j}$ with integral $\tilde{z}^{if}$ represents a feasible intensity assignment to activity j such that j starts not sooner than time $\ell + 1$ (assuming that $z_t^{if} = \tilde{z}_t^{if}$ for $t \in \{r^j, \ldots, d^{if}\}$).

Next, we characterize the members of K_f^{i*} and K_f^{*j}, respectively. First notice that K_f^{*j} is equivalent to K^{*j} (defined in the end of the previous Section), the only difference being that we use z^{if} instead of z^i in the definition. On the other hand, K_f^{i*} can be obtained from the following polytope K_f by reindexing variables, and appropriate substitutions. Let $n \geq m \geq p \geq 1$ be positive integers, $0 < a \leq 1$, $0 < f \leq 1$ positive numbers, and suppose $(p-1) \cdot a < f \leq p \cdot a$ hold. K_f is the convex hull of points $(x, z) \in \mathbb{R}^n \times \{0,1\}^{m-p}$ satisfying the following linear system:

$$\sum_{t=1}^{n} x_t = 1, \tag{5.23}$$

$$\sum_{t=1}^{\ell-1} x_t \geq f \cdot (1 - z_\ell), \quad \ell \in \{p+1, \ldots, m\}, \tag{5.24}$$

$$\sum_{t=1}^{m} x_t \geq f, \tag{5.25}$$

$$z_t \geq z_{t+1}, \quad t \in \{p+1, \ldots, m-1\}, \tag{5.26}$$

$$0 \leq x_t \leq a, \quad t \in \{1, \ldots, n\}. \tag{5.27}$$

In any vertex $(\hat{x}, \hat{z})$ of K_f, $\hat{z}$ is equal to precisely one of the following vectors in $\{0,1\}^{m-p}$:

$$z_t^\ell = \begin{cases} 1, & t \in \{p+1, \ldots, \ell\}, \\ 0, & t \in \{\ell+1, \ldots, m\}, \end{cases} \qquad \ell \in \{p, \ldots, m\}.$$

A generalization of Lemma 5.1 can be easily derived:

LEMMA 5.3 *A vector* $(x, z) \in \mathbb{R}_+^n \times \mathbb{R}_+^{m-p}$ *belongs to* K_f *if and only if the scalars*

$$\lambda_\ell = \begin{cases} 1 - z_{p+1} & \ell = p, \\ z_\ell - z_{\ell+1}, & \ell \in \{p+1, \ldots, m-1\}, \\ z_m, & \ell = m. \end{cases} \tag{5.28}$$

are all non-negative, and there exist vectors $x^\ell \in [0, a]^n$, $(\ell \in \{p, \ldots, m\})$, *such that* $\sum_{t=1}^{\ell} x_t^\ell \geq f$, $\sum_{t=1}^{n} x_t^\ell = 1$, *and* $\sum_{\ell=p}^{m} \lambda_\ell (x^\ell, z^\ell) = (x, z)$.

Analogously to Lemma 5.2, one can prove the following, more general, result:

LEMMA 5.4 $(x^i, x^j, z^{if}) \in K_f^{ij}$ *if and only if* $(x^i, z^{if}) \in K_f^{i*}$, $(x^j, \tilde{z}^{if}) \in K_f^{*j}$, $z_t^{if} = \tilde{z}_t^{if}$ *for* $t \in \{r^j, \ldots, d^{if}\}$, *and* (x^i, x^j, z^{if}) *satisfies ineq. (5.22).*

Therefore, the linear representations of K_f^{i*} and K_f^{*j} together give a linear representation of K_f^{ij}.

5.4.2 A linear representation of K_f

Recall that p is the least integer with $p \cdot a \geq f$. Let $a_r = f - (p-1) \cdot a$.

THEOREM 5.5 *K_f equals the set of vectors (x, z) in $[0, a]^n \times [0, 1]^{m-p}$ that satisfy the following linear constraints:*

$$\sum_{t=1}^{n} x_t = 1,$$

$$a_r z_{t_1} + a \sum_{t \in S_1 \setminus \{t_1\}} z_t + \sum_{t \in \{1,\ldots,t_1\} \setminus (S_1 \cup S_2)} x_t \geq f - a|S_2|, \quad (5.29)$$

for every $\emptyset \neq S_1 \subseteq \{p+1, \ldots, m\}$, $S_2 \subset \{1, \ldots, p\}$ and $|S_1| + |S_2| = p$, t_1 being the greatest element of S_1;

$$(f-1)z_\ell + \sum_{t \in U_\ell^1} a z_t + \sum_{t \in U_\ell^2} a z_\ell + \sum_{t \in \{1,\ldots,n\} \setminus (U_\ell^1 \cup U_\ell^2)} x_t \geq f, \quad (5.30)$$

for $\ell \in \{p+1, \ldots, m\}$, $U_\ell^1 \subseteq \{p+1, \ldots, \ell\}$, $U_\ell^2 \subseteq \{\ell+1, \ldots, n\}$, $a|U_\ell^1 \cup U_\ell^2| \leq 1$, and $a|U_\ell^2| \leq 1 - f$;

$$\sum_{t \in U} (x_t - a z_t) \geq 1 - f, \quad (5.31)$$

for $U \subseteq \{p+1, \ldots, m\}$ with $a|U| \leq 1$.

*Proof*Define a capacitated network $N_{x,z}$ with source s and sink r, and two nodes v_ℓ^1, v_ℓ^2 for each $\ell = p, \ldots, m$, and one node w_t for each $t = 1, \ldots, n$. There is an arc from source s to each node v_ℓ^1 with capacity $c(s, v_\ell^1) = \lambda_\ell$, $\ell = p, \ldots, m$. There is an arc from v_ℓ^1 to each w_t with $1 \leq t \leq \ell$, having capacity $c(v_\ell^1, w_t) = a \cdot \lambda_\ell$, and an arc (v_ℓ^1, v_ℓ^2) with capacity $c(v_\ell^1, v_\ell^2) = \lambda_\ell(1-f)$. v_ℓ^2 is connected to each w_t with $\ell+1 \leq t \leq n$ with an arc of capacity $c(v_\ell^2, w_t) = a \cdot \lambda_\ell$. Finally, each w_t is connected to sink r with an arc of capacity x_t. A fragment of the network is show in Figure 5.1. Using Lemma 5.3, we can show the following result:

LEMMA 5.6 *$(x, z) \in K_f$ if and only if $x \in [0, a]^n$, the λ_ℓ (defined by (5.28)) are all non-negative, and the minimum capacity of an $s - r$ cut in $N_{x,z}$ is 1.*

*Proof*By Lemma 5.3, $(x, z) \in K_f$ if and only if the scalars λ_ℓ, $\ell = p, \ldots, m$, are non-negative and there exist vectors $x^\ell \in [0, a]^n$, $\ell = p, \ldots, m$, such that $\sum_{t=1}^{\ell} x_t^\ell \geq f$, $\sum_{t=1}^{n} x_t^\ell = 1$, and $\sum_{\ell=p}^{m} \lambda_\ell(x^\ell, z^\ell) = (x, z)$. The vectors $\lambda_\ell x^\ell$ give rise to a compatible flow of value 1 in $N_{x,z}$. To see this, let $g(s, v_\ell^1) = \lambda_\ell$, $g(v_\ell^1, w_t) = \lambda_\ell x_t^\ell$ for $1 \leq t \leq \ell$, $g(v_\ell^1, v_\ell^2) = \sum_{t=\ell+1}^{n} \lambda_\ell x_t^\ell$, $g(v_\ell^2, w_t) = \lambda_\ell x_t^\ell$

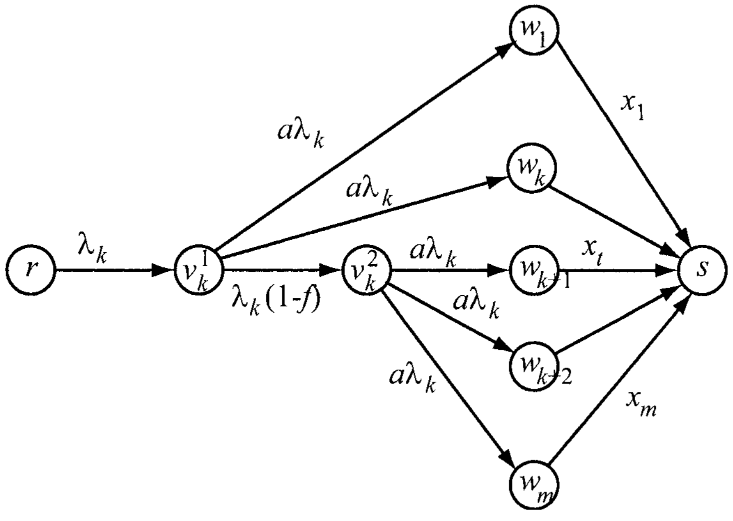


Figure 5.1. A fragment of $N_{x,z}$.

for $t = \ell + 1, \ldots, n$, and $g(w_t, r) = x_t$. Since $\sum_\ell \lambda_\ell = 1$ by definition, $x^\ell \in [0,a]^n$, $\sum_t x_t^\ell = 1$, and $\sum_\ell \lambda_\ell x^\ell = x$ by assumption, it follows that g is a compatible $s - r$ flow in $N_{x,z}$. Finally, note that by the Max-Flow Min-Cut Theorem of Ford and Fulkerson (1956), the value of a maximum $s - r$ flow in $N_{x,z}$ is 1 if and only if the minimum capacity of an $s - r$ cut is 1, from which necessity of the conditions follows.

Conversely, if the conditions of the lemma hold, then there exists a compatible $s - r$ flow g of value 1 in $N_{x,z}$. If $\lambda_\ell > 0$, define x_t^ℓ as $g(v_\ell^1, w_t)/\lambda_\ell$ for $t \in \{1, \ldots, \ell\}$, and $g(v_\ell^2, w_t)/\lambda_\ell$ for $t \in \{\ell + 1, \ldots, n\}$. When $\lambda_\ell = 0$, choose any $x^\ell \in [0,a]^n$ such that $\sum_t x_t^\ell = 1$ and $\sum_{t=1}^{\ell} x_t^\ell \geq f$. This choice of the x^ℓ satisfies the conditions of Lemma 5.3, as one may verify. Therefore, $(x, z) \in K_f$. □

The capacity of an $s - r$ cut $[S, \overline{S}]$ can be expressed as follows. First notice that $s \in S$ and $r \notin S$. In addition, S my contain some of the nodes v_ℓ^1, v_ℓ^2, and w_t, and the rest of the nodes are in $\overline{S}$. The capacity $c([S, \overline{S}])$ of the cut is

$$\sum_{v_\ell^1 \in \overline{S}} \lambda_\ell + \sum_{(v_\ell^1, w_t) \in [S, \overline{S}]} a\lambda_\ell + \sum_{(v_\ell^1, v_\ell^2) \in [S, \overline{S}]} \lambda_\ell(1 - f) + \sum_{(v_\ell^2, w_t) \in [S, \overline{S}]} a\lambda_\ell + \sum_{w_t \in S} x_t.$$

In order to show the validity of the inequalities (5.29) consider the $s - r$ cut with $S = \{s\} \cup \{v_\ell^1 \mid \ell = p, \ldots, t_1 - 1\} \cup \{w_t \mid t \in \{1, \ldots, t_1\} \setminus (S_1 \cup S_2)\}$.

The capacity of this cut can be expressed as follows:

$$c([S,\overline{S}]) = \sum_{\ell=t_1}^{m} \lambda_\ell + \sum_{\ell=p}^{t_1-1} \sum_{t=1}^{\ell} a\lambda_\ell + \sum_{\ell=p}^{t_1-1} \lambda_\ell(1-f) + \sum_{t\in\{1,\dots,t_1\}\setminus(S_1\cup S_2)} x_t.$$

To simplify the above expression, for $t \in \{1,\dots,p\}$ introduce new symbols z_t each of value 1. Observe that $\sum_{\ell=t_1}^{m} \lambda_\ell = z_{t_1}$ and $\sum_{\ell=p}^{t_1-1} \lambda_\ell = 1 - z_{t_1}$. Therefore, we obtain

$$z_{t_1} + \sum_{w_t\in\overline{S},t<t_1} a(z_t - z_{t_1}) + (1-z_{t_1})(1-f) + \sum_{t\in\{1,\dots,t_1\}\setminus(S_1\cup S_2)} x_t.$$

After simplification we get

$$1 - f + a[S_2| + (f-(p-1)a)z_{t_1} + a \sum_{t\in S_1\setminus\{t_1\}} z_t + \sum_{t\in\{1,\dots,t_1\}\setminus(S_1\cup S_2)} x_t.$$

For any $(x,z) \in K_f$, the capacity of the cut $[S,\overline{S}]$ is at least 1, and since $f-(p-1)a = a_r$, our claim is proved.

To show the validity of (5.30), choose $S = \{s\}\cup\{v_k^1, v_k^2 \mid k \in \{\ell,\dots,m\}\}\cup\{w_t \mid t \in \{1,\dots,n\} \setminus (U_\ell^1 \cup U_\ell^2)\}$. As for (5.31), consider $S = \{s\} \cup \{v_p,\dots,v_m\} \cup \{w_t \mid t \in \{1,\dots,m\} \setminus U\}$. The details are omitted.

Conversely, suppose there exists $(x,z) \in P \setminus K_f$, where P is the polytope consisting of the vectors (x,z) satisfying the conditions of the theorem. Since $(x,z) \notin K_f$, by Lemma 5.6 the minimum capacity of an $s-r$ cut in $N_{x,z}$ is strictly smaller than 1. We argue there exists a minimum capacity $s-r$ cut $[S,\overline{S}]$ equivalent to one of the inequalities (5.29), (5.30) or (5.31). Let $W = \{w_1,\dots,w_n\}$.

Clearly, there exists $v_\ell^1 \in S$, otherwise the capacity of $[S,\overline{S}]$ is at least $\sum_{\ell=p}^{m} \lambda_\ell = 1$, a contradiction.

Now suppose there exists $\ell \in \{p,\dots,m-1\}$ with $v_\ell^1, v_\ell^2 \in S$. Then we may add v_k^1, v_k^2 to S for all $k \in \{\ell+1,\dots,m\}$ without increasing the capacity of the cut. Namely, since $v_\ell^1, v_\ell^2 \in S$, the cardinality of the set $U = W \cap \overline{S}$ (the neighbors of v_ℓ^1, v_ℓ^2 not in S) satisfies $a|U| \leq 1$, otherwise removing v_ℓ^1 and v_ℓ^2 from S would yield a cut of smaller capacity which is not possible. One similarly shows that the cardinality of the set $U_\ell^2 = \{w_{\ell+1},\dots,w_n\} \cap \overline{S}$ (the neighbors of v_ℓ^2 not in S) is bounded by $a|U_\ell^2| \leq 1-f$. Consider any $k \in \{\ell+1,\dots,m\}$. If $v_k^1 \in \overline{S}$, then we can add v_k^1, v_k^2 to S without increasing the capacity of the cut, because the set of neighbors of v_k^1, v_k^2 not in S is precisely U. On the other hand, when $v_k^1 \in S$, but $v_k^2 \in \overline{S}$, then adding v_k^2 to S cannot increase the capacity of the cut, since $|U_\ell^2| \geq |U_k^2|$.

Still assuming there exists ℓ with $v_\ell^1, v_\ell^2 \in S$, choose ℓ the smallest with this property. Since $az_t = a$ for all $t \in \{1,\dots,p\}$ and $x_t \leq a$ as $(x,z) \in P$, we may assume that $U \subseteq \{p+1,\dots,n\}$. We claim that for all $k \in \{p,\dots,\ell-1\}$, $v_k^1 \in S$. To the contrary, suppose $v_k^1 \in \overline{S}$ and $k \leq \ell - 1$. If we add v_k^1, v_k^2 to S, the capacity will change by $-\lambda_k + a|U|\lambda_k$, which is at most 0 as we have already seen. Hence, we can set ℓ to k and repeat the above argument. Now we compute the capacity of $[S,\overline{S}]$ under the above conditions. If $\ell = p$, i.e., $v_k^1, v_k^2 \in S$ for all $k \in \{p,\dots,m\}$, then $c([S,\overline{S}]) = a|U|\sum_k \lambda_k + \sum_{w_t \in W\setminus U} x_t = a|U| + \sum_{w_t\in W\setminus U} x_t \geq 1$, a contradiction. Now suppose $\ell > p$. Let $U_\ell^1 = \{w_{p+1},\dots,w_\ell\} \cap \overline{S}$.

$$c([S,\overline{S}]) = (1-z_\ell)(1-f) + \sum_{w_t \in U_\ell^1} a z_t + \sum_{w_t \in U_\ell^2} a z_\ell + \sum_{w_t \in W\setminus U} x_t.$$

Since $c([S,\overline{S}]) < 1$ by assumption, we obtain that (5.30) is violated by (x,z), therefore, $(x,z) \notin P$, a contradiction. (Here we slightly abused notation, by using U_ℓ^1, U_ℓ^2 for denoting sets of nodes and sets of corresponding indices, respectively.)

From now on we assume that $v_\ell^2 \in \overline{S}$ for all ℓ. We claim that if $v_\ell^1 \in S$, then for all $k \in \{p,\dots,\ell-1\}$, $v_k^1 \in S$. Since $v_\ell^1 \in S$, $a|U_\ell^1| \leq f$, otherwise removing v_ℓ^1 from S would decrease the capacity of the cut by $(a|U_\ell^1| + (1-f))\lambda_k - \lambda_k > 0$, which contradicts that $[S,\overline{S}]$ has minimal capacity. Now if $k < \ell$, $|U_k^1| \leq |U_\ell^1|$, therefore, $a|U_k^1| \leq f$, implying our claim.

To summarize, ℓ satisfies that either $p \leq \ell < m$ and $a|U_\ell^1| < f \leq a(|U_\ell^1|+1)$, or $\ell = m$ and $a|U_m^1| \leq 1$. In the former case, $p - 1 = |U_\ell^1|$ must hold by the definition of p. Moreover, letting $t_1 = \ell + 1$, a simple calculation shows that the cut is equivalent to (5.29) with $S_1 = \{t \in \{p+1,\dots,\ell\} \mid w_t \in U_\ell^1\} \cup \{t_1\}$, $S_2 = \{t \in \{1,\dots,p\} \mid w_t \in U_\ell^1\}$. Hence, (5.29) is violated by (x,z) implying that $(x,z) \notin P$, a contradiction. Finally, when $\ell = m$, the cut is equivalent to (5.31), again, a contradiction. No more cases are left, $P \subseteq K_f$ is proved. □

5.5 Preliminary computational results

In order to assess the appropriateness of the MIP formulation and the power of the cutting planes developed in the present paper, we implemented a Branch-and-Cut based solver for our problem, for a general introduction to this method see Padberg and Rinaldi (1987), and Jünger et al (1995).

5.5.1 The Branch-and-Cut algorithm

Branch-and-Cut is a general technique for solving mixed-integer programs. It is a branch-and-bound algorithm using the continuous relaxation of the problem in which all integrality restrictions are dropped. In addition, cuts are generated during the search. A *cut* is an inequality valid for the convex hull of feasible solutions of the original problem, but violated by the (fractional) solution corresponding to a node of the search-tree. When applying the method to a problem, firstly classes of valid inequalities must be identified, and for each class a separation routine must be supplied. Given a (fractional) solution to the problem, the separation routine aims at finding a violated inequality in the class.

Since the inequalities (5.29), (5.30) and (5.31) are valid for K_f, they are valid for K_f^{i*}, too (after reindexing variables and setting p to p^i and m to d^{if}). On the other hand, (5.12) is valid for K_f^{*j}. A separation algorithm for (5.12) is described in Kis (2004). In fact, the same algorithm can be used to separate (5.29). For the sake of completeness, we describe the algorithm for separating the (5.29) inequalities.

By adding $\sum_{t\in S_1\setminus\{t_1\}} x_t + \sum_{t\in S_2} x_t + \sum_{t\in\{t_1,\ldots,n\}} x_t$ to both sides of (5.29) we get:

$$a_r z_{t_1} + \sum_{t\in S_1\setminus\{t_1\}} a z_t + 1 \geqslant f - a|S_2| + \sum_{t\in S_1\setminus\{t_1\}} x_t + \sum_{t\in S_2} x_t + \sum_{t\in\{t_1,\ldots,n\}} x_t.$$

Rewriting gives the following, equivalent inequality:

$$\sum_{t\in S_1\setminus\{t_1\}} (a z_t - x_t) + \sum_{t\in S_2} (a - x_t) \geqslant f - a_r z_{t_1} - \sum_{t\in\{1,\ldots,t_1-1\}} x_t.$$

For a fixed t_1 it is easy to verify whether the above inequality is violated for some $S_1 \subseteq \{p+1,\ldots,t_1\}$ with $t_1 \in S_1$, and $S_2 \subset \{1,\ldots,p\}$, $|S_1|+|S_2| = p$. Namely, define $t_1 - 1$ items $e_1,\ldots,e_{t_1-1}$ with weights $w(e_t) = a - x_t$ if $t \leqslant p$, and $w(e_t) = a z_t - x_t$, when $p+1 \leqslant t \leqslant t_1 - 1$. Take the first $p-1$ smallest weight items, and define the sets S_1 and S_2 with them (t_1 belongs S_1 by definition). Clearly, the sets S_1 and S_2 defined above gives the smallest left hand side. The right hand side can easily be calculated, as it does not depend on S_1 and S_2. Comparing the left and right hand sides, we can decide whether (5.29) is violated for a given t_1.

For a fixed t_1 the calculation takes $O(n \log n)$ time. Surprisingly, the total time needed to verify all possible t_1 can also be done in $O(n \log n)$ time. Notice that we can start with $t_1 = n$, in which case we sort all items $e_1,\ldots,e_{n-1}$. Then, when decreasing t_1, we just modify the left hand side and the right hand side in $O(1)$ time on average, using appropriate data structures.

As for (5.30) and (5.31), these inequalities can be separated by, e.g., solving the maximum flow problem defined in Lemma 5.6 (though it is a slow separation procedure). However, preliminary computations had shown that these inequalities had been violated only very rarely, therefore, in the following tests they were not separated.

In our tests we compared two algorithms: B^+ and B^-. The algorithm B^+ separates the following cuts:

(i) for each K_f^{ij}, the inequalities (5.29) for K_f^{i*}, and the inequalities (5.12) adapted to $K_f^{*j} = K^{*j}$.

(ii) Flow Cover inequalities, as defined by Padberg et al (1985).

(iii) Gomory's fractional cuts, see e.g., Nemhauser and Wolsey (1988).

The cuts (5.29) and (5.12) will be called (S_1, S_2) cuts.

The algorithm B^- separates only the inequalities (ii) and (iii) above.

Both algorithms were implemented in C++ using the ILOG MIP Library ver. 7.5 through the ILOG Concert Technology interface (these are products of ILOG). Each algorithm started with preprocessing that sometimes eliminated hundreds of rows and columns. The inequalities in class (i) were separated by our procedure, while those in classes (ii) and (iii) were separated automatically by the solver. Moreover, we used the built-in heuristic to find feasible solutions during the search.

5.5.2 Test instances

Being not aware of any benchmark instances to the problem at hand, we have modified the instances of De Boer (1998). These instances were devised for the problem with finish-to-start precedence constraints (cf. Section 5.3). The most important parameters are the following:

- number of activities, n,
- number of resources, r,
- average slack, s, which is computed as $s = \sum_{i=1}^{n}(d^i - r^i + 1 - p^i)/n$, where p^i is the minimum time to complete activity i.

The processing times and the precedence relations were generated at random. De Boer obtained a total of 450 instances by generating 10 random instances for each combinations of the above parameters, where $n \in \{10, 20, 50\}$, $r \in \{3, 10, 20\}$ and $s \in \{2, 5, 10, 15, 20\}$. We modified each instance by setting $f_{ij} = 0.5$ for each pair of activities (i, j) connected by a precedence constraint. That is, we replaced each finish-to-start precedence constraint by a feeding precedence constraint with an overlap of 0.5.

5.5.3 Results

We run each of the algorithms B^+ and B^- on each instance for 420 seconds of CPU time on a PC with a 2.4GHz Pentium 4 processor and Windows 2000 operating system. When the time limit was hit, the search terminated and the algorithms returned the best upper and lower bounds found so far. Let ub^+ and lb^+ denote the best upper and lower bounds, respectively, found by algorithm B^+. Similarly, let ub^- and lb^- denote the best upper and lower bounds determined by B^-.

Table 5.1 summarizes the average $ub\,/\,lb$ values for both algorithms. Averages are taken over the ten instances in each class defined by the parameters (n, r, s). For each class there are two numbers, the first one gives the performance for algorithm B^+, whereas the second one gives that of algorithm B^-. A value of 1 indicates that all instances in the class were solved to optimality within the given time limit. As can be seen, algorithm B^+ performs slightly better than algorithm B^-, except in the class with $(n = 50, r = 20, s = 20)$. In fact, in this class B^+ did not obtain a feasible solution for 3 out of the 10 instances within the 420 CPU seconds time limit.

Table 5.1. Average ub/lb values for algorithm B^+ (first row), and B^- (second row), respectively, in each class (n, r, s).

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	3	10	20	3	10	20
2	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1.02	1.06
	1	1	1	1	1	1	1.02	1.03	1.07
20	1	1	1	1	1	1.01	1.01	1.37	1.17
	1	1	1	1	1.01	1.01	1.01	1.5	1.16

Table 5.2 compares the upper and the lower bounds obtained by the two algorithms. Notice that in each class (n, r, s), the first row contains the average ratio ub^+/ub^-, while the second row comprises the average lb^+/lb^- value. When $ub^+/ub^- < 1$, the algorithm B^+ found a better feasible solution than B^- on average. In contrast, when $lb^+/lb^- < 1$, then algorithm B^+ proved a weaker lower bound than algorithm B^- on average. Again, in most cases the two algorithms found the same lower and upper bounds, and in a few classes B^+ outperformed B^-, while the converse essentially happened only in the class $(n = 50, r = 20, s = 20)$.

Finally, Tables 5.3 and 5.4 summarize the most important aspects of the computations with algorithm B^+ and B^-, respectively. For B^+ we provide the average CPU time (in seconds), the average number of search-tree nodes, and the average number of Flow Cover, Gomory's fractional and the (S_1, S_2) cuts generated over the ten instances in each class (n, r, s). For the algorithm B^-, only average CPU times and average number of search-tree nodes are provided.

Table 5.2. Average ub^+/ub^- (first row), and lb^+/lb^- (second row).

	$n = 10$			$n = 20$			$n = 50$		
s	$r = 3$	10	20	3	10	20	3	10	20
2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
5	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
10	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
15	1.000	1.000	1.000	1.000	1.000	1.000	0.981	0.999	0.994
	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.001	1.001
20	1.000	1.000	1.000	1.000	0.996	1.000	0.998	0.952	1.010
	1.000	1.000	1.000	1.000	0.999	0.999	1.003	1.013	0.998

Observe that adding our problem specific cutting planes is not always advantageous. The difference between the performance of the two algorithms is minor. In contrast, for the special case with finish-to-start precedence constraints treated in Kis (2004), there was a clear cut between the results obtained with and without problem specific cutting planes: it was evident that by adding problem specific cuts we got better results. A partial explanation can be the differences between the MIP formulations of the two problems:

- The formulation for the problem with finish-to-start precedence constraints contains the inequalities $x_t^j \leq a^j z_t^j$. These inequalities are not valid for the problem with feeding precedence constraints.
- The inequalities $\sum_{t=r^i}^{\ell-1} x_t^i \geq f_{ij}(1-z_\ell^i)$ in the formulation of the problem with feeding precedence constraints are strong enough for the MIP solver to find good solutions quickly.

Note however that we also run the two algorithms presented in this paper on the instances with finish-to-start precedence constraints and found that the results were inferior to those obtained in Kis (2004) using a different MIP formulation and problem specific cuts. On the other hand, we got better results in terms of the ratio of the upper and lower bounds for the problem with feeding precedence constraints. This hints that project scheduling with feeding precedence constraints is an easier problem than that with finish-to-start precedence constraints.

5.6 Conclusions

In this paper we have investigated an extension of RCPSP with variable intensity activities and feeding precedence constraints. We have devised a MIP

Table 5.3. Averages of CPU time, search-tree nodes, Flow Cover, Gomory's fractional and (S_1, S_2) cuts for algorithm B^+.

s	$n = 10$			$n = 20$			$n = 50$		
	$r = 3$	10	20	$r = 3$	10	20	$r = 3$	10	20
	0.034	0.028	0.017	0.023	0.025	0.027	0.036	0.055	0.067
	0.100	0.500	0.200	0.300	0.300	0.500	0.400	0.900	1.100
2	0.200	1.300	0.800	0.700	0.800	1.600	1.000	2.100	4.700
	0.900	1.000	1.000	1.700	2.400	2.100	4.100	4.300	6.400
	0.200	0.200	0.000	0.900	0.600	0.400	1.000	1.300	1.600
	0.039	0.048	0.049	0.108	0.164	0.128	0.216	0.856	1.201
	0.800	2.600	1.700	2.600	8.900	8.000	5.000	27.100	63.300
5	1.500	1.900	3.900	2.800	7.500	4.800	1.800	8.900	18.800
	3.000	3.900	7.000	6.500	9.700	10.200	8.300	12.300	18.400
	1.600	1.500	1.300	5.700	6.600	3.900	6.300	12.400	14.100
	0.156	0.247	0.211	0.430	2.222	1.561	7.975	34.712	171.150
	5.500	15.400	17.100	7.100	106.100	139.800	81.800	395.300	1244.900
10	2.600	7.300	8.300	2.100	18.400	19.200	3.500	24.100	50.500
	8.700	7.400	12.100	5.000	16.100	20.500	11.600	17.500	25.300
	6.500	7.100	5.300	15.900	28.800	19.900	44.500	84.400	161.800
	0.853	0.944	1.085	1.886	14.456	51.381	85.616	389.870	420.067
	39.900	56.600	96.900	12.700	409.300	2247.200	118.500	535.500	511.000
15	5.200	13.800	12.300	4.800	22.600	44.200	3.800	42.300	75.400
	3.600	12.400	15.600	8.500	16.100	29.600	12.300	14.100	24.800
	26.700	17.100	15.700	22.700	99.200	119.400	111.300	281.900	387.700
	1.386	4.328	2.011	11.133	97.577	212.791	152.067	420.080	420.113
	30.900	175.600	115.400	104.600	727.200	2734.300	151.400	202.857	164.375
20	7.200	22.000	15.200	6.700	39.700	50.000	5.300	54.429	85.875
	6.400	17.200	19.200	5.700	15.400	33.000	15.500	17.286	29.000
	22.400	34.400	31.400	60.400	176.400	215.000	165.700	288.143	312.000

Table 5.4. Average CPU time (first row) and search-tree nodes (second row) for algorithm B^-.

s	$n = 10$			$n = 20$			$n = 50$		
	$r = 3$	10	20	$r = 3$	10	20	$r = 3$	10	20
2	0.03	0.03	0.02	0.02	0.02	0.02	0.03	0.04	0.05
	0.10	0.50	0.20	0.60	0.40	0.20	0.40	0.50	0.70
5	0.03	0.04	0.05	0.08	0.16	0.11	0.19	0.84	1.05
	1.00	1.50	2.40	4.30	11.80	6.30	5.60	31.90	65.20
10	0.13	0.21	0.20	0.44	1.90	1.77	9.65	37.52	164.69
	5.40	16.60	14.20	11.90	108.50	191.50	144.70	474.10	1358.50
15	0.64	0.84	0.96	1.59	13.85	61.81	90.78	384.43	420.07
	39.00	65.40	99.80	16.00	547.10	3286.00	154.00	776.10	638.70
20	1.24	3.49	1.86	12.88	95.34	207.31	201.76	420.10	420.09
	32.40	176.00	104.60	146.80	921.90	4130.70	316.10	177.00	168.90

formulation that proved effective in a branch-and-cut approach. We have also obtained new polyhedral results that can be used to strengthen the formulation.

The model with feeding precedence constraints seems easier to solve than that with finish-to-start precedence constraints. Finding the core of this phenomenon is subject to future research.

Another way of extending this work is to devise new cutting planes that reduce the computation time significantly.

Acknowledgments

The author is grateful to the anonymous referee whose comments helped to improve the presentation.

References

De Boer, R. (1998). Resource-Constrained Multi-Project Management - A Hierarchical Decision Support System, Ph.D. thesis, Twente University Press, The Netherlands.

Ford, L. R., and Fulkerson, D. R. (1956). Maximal flow through a network, *Canadian Journal of Mathematics* 8:399–404.

Gademann, N., and Schutten, M. (2005), Linear programming based heuristics for project capacity planning, *IIE Transactions*, 37(2):153–165.

Hans, E. W. (2001). Resource loading by branch-and-price techniques, Ph.D. thesis, Twente University Press, The Netherlands.

Jünger, M., Reinelt, G., and Thienel, S. (1995). Practical problem solving with cutting plane algorithms in combinatorial optimization, *DIMACS Ser. in Discr. Math. and Theor. Comput. Sci.* 20:111–152.

Kis, T. (2004). A branch-and-cut algorithm for scheduling of projects with variable intensity activities, *Math. Prog.*, in press.

Leachman, R. C. (1983). Multiple resource leveling in construction systems through variation of activity intensities, *Naval research logistics quarterly* 30(3):187–198.

Leachman, R. C., Dincerler, A., and Kim, S. (1990). Resource-constrained scheduling of projects with variable-intensity activities, *IIE Transactions* 22(1):31–39.

Márkus, A., Váncza, J., Kis, T., and Kovács, A. (2003). Project oriented view on production planning, *CIRP Annals of Manufacturing Technology* 52(1):359–362.

Nemhauser, G. L., and Wolsey, L. A. (1988) *Integer and Combinatorial Optimization*, John Wiley & Sons, New York.

Padberg, M. W., Van Roy, T. J., and Wolsey, L. A. (1985), Valid linear inequalities for fixed charge problems, *Operations Research* 33(4):842–861.

Padberg, M. W., and Rinaldi, G. (1987). Optimization of a 532 city symmetric traveling salesman problem by branch and cut, *Oper. Res. Lett.* 6:1–7.

Tavares, L. V. (1998). *Advanced models for project management*, Kluwer Academic Publishers, pp. 177-216.

Tavares, L. V. (2002). A review of the contribution of Operational Research to Project Management, *Eur. Jour. Ops. Res.* 136:1–18.

Weglarz, J. (1976). Time-optimal control of resource allocation in a complex of operations framework, *IEEE Trans. Systems, Man and Cybernetics* 6:783–788.

Weglarz, J. (1979). Project scheduling with discrete and continuous resources, *IEEE Trans. Systems, Man and Cybernetics* 9:644–650.

Weglarz, J. (1981). Project scheduling with continuously-divisible doubly constrained resources, *Management Science* 27(9):1040–1053.

Wullink, G. (2005). Resource loading under uncertainty, Ph.D. Thesis, Twente University Press, The Netherlands.

Chapter 6

MODELLING SETUP TIMES IN PROJECT SCHEDULING

Marek Mika,[1] Grzegorz Waligóra,[1] and Jan Weglarz[1,2]

[1]*Poznań University of Technology, Institute of Computung Science, Piotrowo 2, 60-965 Poznań, Poland;* [2]*Poznań Supercomputing and Networking Center, Noskowskiego 10, 61-704 Poznań, Poland*

Abstract In this chapter project scheduling problems with setup times are considered. Some practical applications justifying considering setups separately from activities are described. An extensive classification of setup times adapted from machine scheduling is proposed, including activity vs. class setup, separable vs. inseparable setup, as well as sequence-independent and sequence-dependent setup times. A new category of setup times - schedule-dependent ones - is discussed. The main part of the paper shows how to model setup times in the presence of particular project components, such as: precedence constraints, resource availability constraints, multiple resource units requests, multiple resources, etc. Some possible extensions of the presented models are given.

Keywords: project scheduling; setup; setup time; setup cost.

6.1 Introduction

In many production processes, where a set of jobs is assigned to a facility to be processed in a given sequence, the facility has to be shut down after finishing one job to prepare it for the next job in the sequence. The operations between processing two consecutive jobs are generally referred to as *setups* or *changeovers*. In this context, these two terms are used synonymously. In general, a setup includes work to prepare a machine, process, or bench for product parts or the cycle (Allahverdi et al (1999)). This includes, but is not limited to: obtaining tools, positioning work in process material, return tooling, cleanup, fastening required jigs and fixtures, adjusting tools or machines, loading and unloading work pieces, inspecting material, transporting resources of different types, and so on. The time and cost required by setup operations are known as *setup time* and *setup cost,* respectively. The costs per unit time for a setup may not be the same for different machines. However, if they are identical for all machines then minimizing the setup cost is equivalent to minimizing the total setup time.

Over the years, in most of the classical machine scheduling research setup times have been considered negligible or as parts of the processing times of jobs. However, while this may be justified for some scheduling problems, in many practical applications this simplifying assumption may lead to solutions of poorer quality, in contrast to solutions obtained by considering setups separately from processing of jobs. Many practical situations require explicit treatment of setups. Moreover, from the point of view of productivity, the jobs requiring the same setup should be processed simultaneously or consecutively in order to reduce the setup time. However, it may not only increase work-in-process inventory but may make the delivery of completed goods late. This trade-off is a real challenge since reducing setup time to zero is impossible even by applying a flexible manufacturing system (FMS). Therefore, the research on production scheduling problems with setup times has grown significantly during the last three decades (Yang and Liao (1999)).

While setup times have been widely considered in machine scheduling, in particular in shop environments (see e.g. Allahverdi et al (1999) or Yang and Liao (1999) for a survey), there is not much literature concerning setup times in the context of project scheduling, although the interest in that area grows. The first publication dealing with setup times in project scheduling was (Kaplan (1991)), since then only a few papers on that subject have been published. They are named and shortly commented in a literature review given in Section 16.4 of this paper.

There are numerous examples of setup times in project scheduling problems. They include situations where resource units must be adjusted or transported in order to perform some activities. Adjusting may mean equipping, resetting, changing, positioning, cleaning, warming up, etc. Examples given in (Kolisch (1995)) include construction projects, where excavators need to be equipped with different types of scoops to perform certain classes of activities, or heavy equipment in general (like trucks, cranes, excavators) might be requested at different construction sites - the operation of moving them from one site to another can be viewed as a setup, etc. Generally, in project scheduling a setup has been defined as a preparation of all the required resources for processing an activity. The time needed for this preparation is then called a setup time. However, the preparation must be, in fact, considered with respect to particular resource units rather then resources alone. Indeed, the resource units of a particular resource have the same functionality, and the preparation of each of them for processing a given activity is identical. Nevertheless, a setup has to be performed on each unit separately, and - even more importantly - the activity must use exactly these resource units that are prepared for processing it, i.e. the ones on which setups have been done. The remaining units of the given resource are simply not ready for processing the activity, since setups have not been performed on those units. Of course, it is possible to treat each resource unit of a particular resource as a separate resource itself, but it is not justified since, as it has been mentioned, they have the same functionality. As a result, in project scheduling setups have to be considered in reference to resource units, not just resources as a whole.

In the area of project scheduling, the assumption that the duration of an activity reflects both setup and processing times has also been made throughout the years. This common assumption can be justified as long as setup times are relatively small in comparison to processing times. But if some activities require the same considerable setup, the makespan can be reduced by batching these activities. In this case, modelling and solving such a problem as a classical project scheduling problem may lead to poor solutions (Kolisch (1995)). Moreover, there are some practical situations where the setup of a given resource for the particular activity can be performed simultaneously with the execution of direct predecessors of this activity. In such a case, the makespan of the project may be decreased at least by the value of the setup time. On the other hand, the concept of treating setups as separate activities has a serious drawback. It follows from the fact that precedence constraints can not be used to model such a situation. Let us consider an example (Figure 6.1) where there is one scarce resource r, two precedence-unrelated activities i and j which are to be executed on resource r, and each of the two activities requires some preparation (setup) of resource r for its execution. The setup for activity i is denoted as s_i, and for activity j as s_j. Now, there are only two possible feasible schedules: $s_i - i - s_j - j$ or $s_j - j - s_i - i$. However, treating setups s_i and s_j as separate activities, and defining precedence relations as $s_i \rightarrow i$ and $s_j \rightarrow j$, incorrect schedules may occur such as: $s_i - s_j - i - j$, $s_i - s_j - j - i$, $s_j - s_i - i - j$ or $s_j - s_i - j - i$ (Figure 6.1). These schedules are precedence- and resource-feasible but are not correct since in each case one of the activities can not be performed because the resource is not prepared for its execution.

The motivation of this paper is to show how to model different types of setup times in static, discrete, and deterministic project scheduling. The remainder of the paper is organized as follows. In Section 16.2 an extensive classification of setup times in project scheduling is presented. In Section 16.3 the definitions of the classical project scheduling problems, i.e. unconstrained, resource-constrained, and multi-mode resource-constrained problems, are presented. Section 16.4 contains a literature review on setup times and costs in project scheduling. Section 16.5 is to show how some project components influence setups. In this section we discuss how to model setup times in various project scheduling problems, taking into account the classification given in

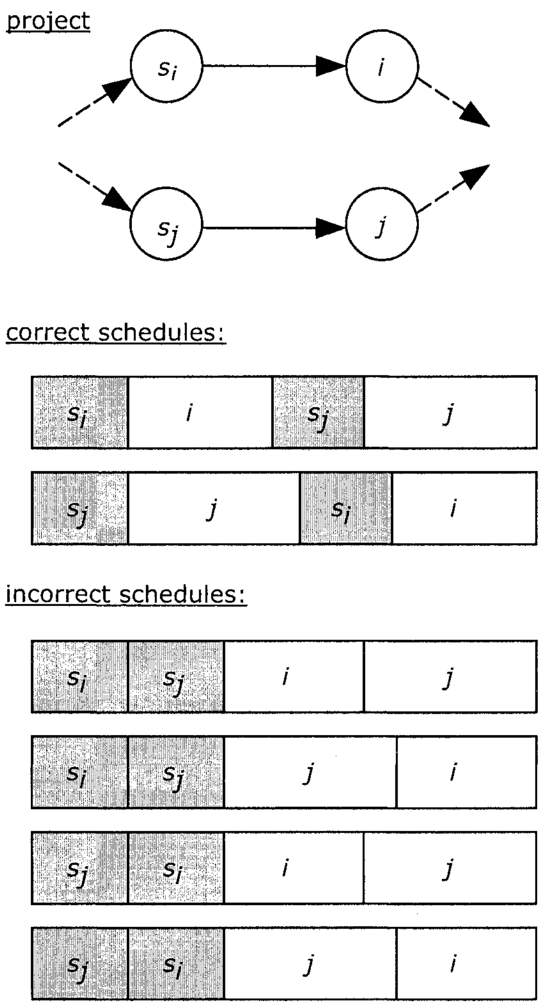


Figure 6.1. Setups treated as activities - an example.

Section 16.2. In Section 15.6 possible extensions of the presented models are commented. Finally, some conclusions are given in Section 15.7.

6.2 Classification of setup times

In this section a classification of setup times adapted from machine scheduling problems is presented. Firstly, the difference between activity setup and class setup is explained. Next, two other types of setups, namely separable and inseparable, are presented. And finally, both sequence-independent and sequence-dependent setup times are discussed, as well as a new category denoted as the schedule-dependent setup time is introduced.

6.2.1 Activity vs. class setup

A setup needed for a resource to switch between two consecutive activities is referred to as the *activity setup.* However, sometimes activities can be grouped into distinct classes according to the similarity of their operations so that all activities in a particular class need the same or similar setup. In other words, the same setup is needed for performing a class of activities. A setup required to change over from one class to another is referred to as the *class setup* (Yang and Liao (1999)). The class setup is also known as the batch or family setup.

Let us recall the previously discussed example with excavators. If this heavy equipment is requested by several activities executed at different construction sites and each such activity requires a different type of scoop, then each setup has to be considered separately as an activity setup. On the other hand, if there

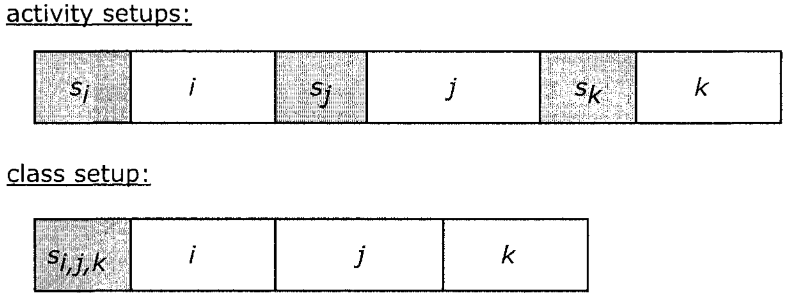


Figure 6.2. Activity and class setups.

are some activities executed at the same site using the same type of scoop, and they can be executed sequentially respecting the precedence and resource constraints, then an excavator may be set up once before the execution of the first activity at the considered construction site. This activity belongs to the class of activities performed with the same settings of resources (the excavator in the considered example). It is not justified to execute any other activity within the sequence of activities from a given class, because it implies at least double execution of some work associated with the setup of a given resource.

6.2.2 Separable vs. inseparable setup

In general, a setup does not have to be performed directly before the activity requiring this setup, there can be other activities or setups performed in between on this resource, provided that none of them destroys the settings required for the relevant activity. Sometimes, before a new activity (class) is ready to be processed on a resource, the setup required for this activity (class) may be completed in advance. In this case, we have the so-called *separable setup*. Otherwise, when a setup must not be performed in anticipation, we call it the *inseparable setup*. In other words, in the case of the inseparable setup, an activity must be started immediately after its setup is finished, whereas in the case of the separable setup any slack greater than zero is allowed between the end of the setup and the start of the corresponding activity. As an example of the

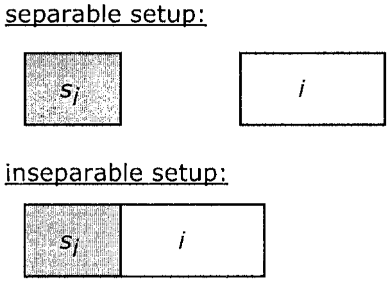


Figure 6.3. Separable and inseparable setups.

resource for which the separable setup can be applied, the previously considered heavy equipment is used. If the setup consists in moving the requested resource from one construction site to another, then it can be done some time periods before starting the corresponding activity in a new location to which this heavy equipment has been moved. So, there is a slack between performing the setup and the activity.

Inseparable setups often occur when some chemical ingredients have to be poured into a grinding, blending, or mixing machine, in order to set up this machine. For example, if an activity is to prepare concrete using a concrete-mixer then a setup of this machine can be to pour into it materials of which the concrete is composed. Sand, gravel, and water can be poured into the concrete-mixer at any time period, but cement and some other specific ingredients have to be poured into this machine just before starting it. So, the process of setting up the concrete-mixer for preparing concrete can be treated as inseparable setup.

6.2.3 Sequence-independent, sequence-dependent, and schedule-dependent setup times

In the next two subsections, two categories of setup times already known from the literature: sequence-independent and sequence-dependent are discussed, as well as a new category called the schedule-dependent setup time is defined.

6.2.3.1 Sequence-independent and sequence-dependent setup times.

There have been two types of setup times considered in the project scheduling literature so far: *sequence-independent setup times*and *sequence-dependent setup times*. In the first case (Kolisch (1995)), setup times depend only on the activity and the resource on which the activity will be processed, but do not depend on the sequence of activities on a particular resource. This case is typical for situations when, e.g., a machine has to be set up for processing some activity but the time needed for this setup is fixed and independent of which activity has been processed previously on this machine. An examples of such situations are given, e.g., in (Conway et al (1967)), where the process of producing different colours on the same machine is considered, and setup times occur for cleaning the machine. In (Bruno and Downey (1978); Monma and Potts (1989); Chen (1993)) a computer system application is described in which computer jobs require different compilers. A setup is not incurred if the next job requires a compiler that is already resident in memory, but if the next job requires a non-resident compiler then a setup time is incurred which depends only on the time needed to load the new compiler. Both abovementioned examples refer to the class setup. Of course, there are numerous examples of sequence-independent setup times referring to activity setup. For example, the time necessary to mounting a material or tool in a mount of a machine tool, as well as time for loading a batch into blast furnace in a steel factory, or pouring cement, water, gravel, sand and other ingredients of concrete into concrete-mixer can be treated as sequence-independent setup times. These examples are activity setups, because, e.g., ingredients used to prepare the concrete or alloy can not be used once more for preparing another concrete or alloy. Several similar examples for sequence-independent setup times can be given as well.

In the second case (Neumann et. al (2003)), setup times depend not only on the activity and the resource, but also on the sequence of activities processed on this resource. For example, if there are three activities i, j, k such that i, j are direct predecessors of k, which are to be processed on the same resource, the setup time needed for processing activity k may be different, depending on whether activity k is executed immediately after activity i (i.e. the sequence is (j, i, k)) or immediately after activity j (i.e. the sequence is (i, j, k)). Such situations are typical, e.g., in industry processes, where processing units, like reactors or filters, have to be cleaned after the completion of certain activities.

In general, the cleaning time will be larger when switching from a low- to a high-quality product then vice versa. Similar situation appears in the context of research and development projects, where the resources represent staff of the same skill working jointly on specific tasks in different interdisciplinary teams, and each task requires a certain lead time necessary to the training of the staff. Since the training of a person moving from one task to another depends on the difference between the two tasks, the lead times and thus the setup times between tasks are sequence-dependent (Neumann et. al (2003)). A further application of project scheduling with sequence-dependent changeover times is given in (Schwindt (2005)). Here, several subprojects sharing common resources are performed in parallel at different sites. The time needed to transfer resource units from one site to another clearly depends on both locations. Such a transfer is required each time a resource unit consecutively processes two activities belonging to different subprojects. Thus, the transfer times can be modelled as sequence-dependent setup times between activities. Generally, sequence-dependent setups are typical for the situations with multi-purpose resources. Other examples include (i) auditing of clients at several places or giving classes and lectures at schools and universities in rooms that are located far apart - a problem of audit-staff scheduling, e.g., was considered in (Dodin and Elimam (1997)), (ii) chemical compounds manufacturing, where the extent of the cleansing depends on both the chemical most recently processed and the next chemical to be processed, (iii) printing industry, when the cleaning and setting of the presses for processing the next job depend on its difference from the colour of ink, size of paper, and the types used in the previous activity, (iv) many other industrial systems like: stamping operations in plastics manufacturing, die changing in metal processing, roll slitting in paper industry, and so on.

6.2.3.2 Schedule-dependent setup times. However, a more general situation than described in the previous subsection is also possible, where setup times do not depend only on the sequences of activities on particular resources, but - more generally - on the assignment of resources to activities over time. It means that not only activities and their sequences on resources affect setup times, but also the fact on which resources predecessors of particular activities have been scheduled. In such a case, the setup times are not just sequence-dependent, but they are *schedule-dependent* (Mika et al (2003)). Let us now explain such situations in more detail.

Assume that we have a classical situation when a set of activities are to be performed on a set of resources. The activities apply for the resources to be executed. Let us call the resources capable of executing different activities *multi-purpose resources.* Apart of multi-purpose resources, there may exist resources dedicated for particular activities, more precisely, resources needed to prepare multi-purpose resources for executing activities. These are resources which the activities do not apply for, they are only necessary for executing particular activities on multi-purpose resources. They can be available from the beginning, or they can be produced by some activities. Let us call such resources *setup-required resources* since they are, in fact, needed for setting up multi-purpose resources for executing activities.

As an example, let us consider a problem of scheduling workflow applications on a computational grid. The problem consists in assigning grid resources to tasks of a workflow job across multiple administrative domains. Workflow applications can be viewed as complex sets of precedence-related various transformations (tasks) performed on some data. The precedence constraints between tasks usually follow from data flow between them, i.e. data files generated by one task are needed to start another task. These data files can be treated as setup-required resources since a node (resource unit) is prepared for processing a task only when all input files required for this task are stored on the local discs of this node. As a result, setup-required resources (data files) are only used for preparing multi-purpose resources (nodes) for executing tasks of a workflow, but tasks do not compete for data files - it is known in advance which data files are dedicated to which tasks, and these files can not be replaced by any other data files. Now, as it is easy to see, the transmission times of these files depend on which nodes the files are transferred between, and the transmission times define the times of multi-purpose resource preparation, i.e. setup times. In other words, the transmission times (setup times) depend not only on the node to which the files are transferred, but also on the nodes where they are located, i.e. on the locations where the preceding tasks have been executed and from which they will be transferred. In consequence, setup times in this problem do not depend only on the sequences of tasks on particular resource units, but - more generally - on the assignment of nodes to tasks over time, so they are schedule-dependent. For more detailed description of the presented example see (Mika et al (2003)).

Thus, schedule-dependent setup times occur when setup-required resources have to be transported or transmitted from one location to another in order to prepare a resource unit in the latter location for executing an activity. The setup-required resources may be available in some locations from the beginning, or they can be an outcome of executing some activities, usually preceding activities, mostly direct predecessors. In consequence, the times needed for providing the setup-required resources (i.e. setup-times) depend on where (in which location, i.e. on which resource unit) the activities producing those resources have been executed, and when. Thus, the setup times clearly depend on the schedule and therefore we call them *schedule-dependent.*

Examples from other areas can be given as well. For instance, an enterprise has several geographically distributed plants and makes products according to the assembly-to-order, make-to-order, or engineer-to-order scenarios. Resources of the enterprise are distributed among the plants, so that some resources may not be available in each plant. In consequence, it is not possible to complete the whole production process within one plant, and specific plants make semi-finished products only which have to be transported between some plants. The semi-finished products are setup-required resources and are dedicated to particular production processes making final products. The transportation times can be then clearly treated as setup times, and depend on the whole production schedule of semi-finished and final products over the distributed plants. One can notice some similarity between the schedule-dependent setup times, especially in the context of the presented grid example, and scheduling with communication delays. The study of scheduling problems with interprocessor communication delays started to be a new research area some years ago, interest in which was rapidly increasing with the development of distributed memory computers. In fact, to get good performance from such computers, the best compromise between parallelism and communication delays has to be found. For a given task graph to be executed, this problem is a scheduling problem where, in addition to the constraints of a classical scheduling problem, communication times between dependent tasks assigned to distinct processors must be taken into account. For a comprehensive survey on scheduling with communication delays see (Chrétienne and Picouleau (1995)).

Let us recall briefly the formulation of the problem of scheduling nonpreemtable, dependent tasks on parallel, identical processors with communication delays. In this problem, each task has to be assigned a processor, and with each precedence relation of the form $i \longrightarrow j$ (which means that task i must be finished before the start of task j) a communication time c_{ij} is associated, where $c_{ij} = 0$ if tasks i and j are executed on the same processor, and $c_{ij} > 0$ otherwise. The objective is the minimization of the makespan.

One of the main classifications for these problems is based on the fact whether task duplication is allowed or not. Since task duplication is not a realistic assumption in the context of project scheduling, we will only consider problems with no duplication. For such problems, some similarity between communication delays and schedule-dependent setup times can be observed. Let us now comment on this issue.

Firstly, let us distinguish between project scheduling and machine scheduling. In project scheduling an activity (a task) can require several units of a given resource, whereas in machine scheduling a task requires only a machine, i.e. one resource unit. However, even if we limit our project scheduling problems to ones with single-unit resource requests, an important difference still occurs, making our problem much more general. It consists in the fact that in the case of communication delays, they can only take two values: 0 or c_{ij}, i.e. if there is a communication delay following from executing precedence-related tasks on distinct processors, its value is always the same. This is not the case in our problem, where schedule-dependent setup times can take many different values, depending on which resource units the successive tasks have been executed. It generally depends on the locations of those resource units, e.g. in the grid example the bandwidth between two nodes, between which data files are transmitted, defines the transmission time and, consequently, the setup time. As a result, we can state that schedule-dependent setup times are a generalization of communication delays already known from the literature.

6.2.4 Removal times

Removal operations are operations which always occur after completing the corresponding activity. Although these operations are executed quite often in practice, they are rarely taken into account in scheduling problems.

There are numerous examples of removal operations like: disengaging tools for a job, releasing a job from jigs and fixtures, dismantling fixtures, jigs and tools, inspecting and sharpening the tools, returning tools to the central depository, cleaning machines and adjacent areas, etc.

Removal operations similarly to setups can be separable or inseparable, and activity or class related. Although removal times can be sequence-independent, sequence-dependent, or schedule-dependent, it is assumed that only sequence-independent removal times are explicitly taken into account. The sequence-dependent as well as schedule-dependent removal times are not considered explicitly, because they are usually included in the setup time of the next activity, that uses the same resources.

When inseparable removal operations are considered for activities, then the removal and processing times of an activity can be aggregated and, in consequence, the removal times can be ignored.

The separable and sequence-independent removal times can be modelled using time lags.

6.3 Classical project scheduling problems

In this section we define the classical unconstrained, resource-constrained, and multi-mode resource-constrained project scheduling problems. The problems are defined mathematically, and all the parameters are presented.

6.3.1 Unconstrained project scheduling problem

The classical unconstrained project scheduling problem (PSP) can be defined as follows. The project consists of a set V of v non-preemptable and precedence-related activities, and a set E of precedence constraints between activities. The strict finish-to-start precedence constraints with zero minimum time lags between some pairs of activities are defined by relations of the type: $i \longrightarrow j$, where $i \longrightarrow j$ indicates that activity i must be finished before the start of activity

j. The structure of the project is represented by a so-called *activity-on-node* (AoN) graph $G(V, E)$ (an alternative representation is called *activity-on-arc* AoA) which is a directed, acyclic, and transitively reduced graph, where the set of nodes V corresponds to the set of activities, and the set of arcs $E = \{(i, j) : i, j \in V; i \rightarrow j\}$ represents the finish-to-start precedence constraints with zero minimum time lags. It is assumed that the nodes of graph G are topologically ordered, i.e. a node has always a higher number than all its predecessors. Moreover, it is assumed that there is exactly one starting and exactly one finishing node in the graph. If this condition is not fulfilled (i.e. there are more than one starting and/or finishing nodes), some special additional nodes representing so-called *dummy activities* have to be inserted at the start and/or at the end of the graph. These dummy activities are usually called the *supersource* and the *supersink*, respectively. The processing times dummy activities are equal to 0, and the supersource has no predecessors whereas supersink has no successors. Each activity of the project is characterized by its processing time, and precedence relations with other activities. The processing time (duration) of activity j is denoted by p_j. The precedence relations of activity j with other activities may be defined by two sets: a set $Prec_j$ of direct predecessors of activity j, and a set $Succ_j$ of direct successors of activity j. Once started, an activity may not be interrupted, i.e. preemption is not allowed. It is assumed that all activities are available from the start of the project. The objective is to find a schedule S that minimizes the project duration (makespan) satisfying all precedence constraints. A schedule assigns a start time S_j to each activity $j \in V$.

It has been proved that the project scheduling problem in the absence of resource constraints can be solved to optimality in polynomial time, using the Critical Path Method (CPM - Kelley (1961)). Thus, no heuristic approaches need to be develop in this case.

Using the classification proposed by Herroelen et al (1999), the defined problem is denoted as $cpm|C_{max}$.

The PSP parameters are summarized in Table 6.1. All information on the project is assumed to be deterministic and known in advance. All numerical parameters of the project are assumed to be non-negative and integer valued.

6.3.2 Resource-constrained project scheduling problem

In the resource-constrained project scheduling problem (RCPSP) activities have to be executed using some scarce resources. Thus, there is additionally a set K_R of R *renewable resources* which are used to execute all activities of the project. Renewable resources are available at any time in limited numbers of units, i.e. an available amount of such a resource is renewed from period to period. The number of units of renewable resource $k, k = 1, ..., R$, available at

Table 6.1. Notation for the PSP.

Symbol	*Definition*
$G(V, E)$	directed AoN graph representing the structure of the project
V	set of activities
$v = \|V\|$	number of activities
$i \rightarrow j$	precedence constraint between activities i and j
E	set of precedence constraints between activities
$Prec_j$	set of direct predecessors of activity j
$Succ_j$	set of direct successors of activity j
p_j	processing time of activity j
$S = \{S_1, S_2, ..., S_v\}$	schedule
S_j	starting time of activity j
C_j	completion time of activity j

each time period is R_k. Activity j requires r_{jk} units of resource $k \in K_R$ in each period of its processing. As a result, each activity of the project is characterized by its processing time, resource requests, and precedence relations with other activities. All resources, as well as activities, are available from the start of the project. The objective is to find a schedule S that minimizes the project duration (makespan) satisfying all precedence and resource constraints. The RCPSP is strongly NP-hard, as a generalization of the well-known job shop problem (Błażewicz et al, 1983).

Using the classification proposed in (Herroelen et al (1999)), the defined problem is denoted as $m, 1|cpm|C_{max}$, and $PS|prec|C_{max}$ according to the classification by Brucker et al (1999).

The additional parameters for the RCPSP are summarized in Table 6.2.

Table 6.2. Additional parameters for the RCPSP.

Symbol	*Definition*
K_R	set of renewable resources
$R = \|K_R\|$	number of renewable resources
R_k	number of units of renewable resource k available in each time period
r_{jk}	request for resource k by activity j

6.3.3 Multi-mode resource-constrained project scheduling problem

The classical RCPSP assumes that each activity can only be executed in a single way which is determined by a fixed duration and fixed resource requests. But there are many practical situations in which the duration of an activity can be decreased at the expense of providing additional resources. In such a situation, an activity may be executed in one of several modes. A mode represents alternative ways of execution of an activity, and is a combination of an activity duration and resource requests. In such a case, the resulting problem is called the multi-mode resource-constrained project scheduling problem (MRCPSP). The main differences between the MRCPSP and the RCPSP are such that for each activity $j \in V$, a set M_j of possible execution modes is defined, and two additional categories of resources: *non-renewable* and *doubly constrained* are introduced. Let us remind that the availability of a renewable resource is once defined and renewed from period to period. For non-renewable resources, total consumption of the resource units is limited for the entire project. In the case of doubly constrained resources, both total and per period availabilities are limited. However, under discrete resources, the doubly constrained resources need not be taken into account explicitly since they can be incorporated by proper enlarging the sets of the first two categories of resources. Thus, we assume an additional set K_N of N *non-renewable resources* which can be consumed by the activities of the project. The number of available units of non-renewable resource $l, l = 1, ..., N$, is N_l. The duration of activity $j \in V$ executed in mode $m_j \in M_j$ is denoted by p_{jm}. Moreover, activity $j \in V$ executed in mode $m_j \in M_j$ requires for its processing r_{jmk} units of renewable resource $k, k = 1, , R$, and n_{jml} units of non-renewable resource $l, l = 1, , N$. A mode chosen for an execution of an activity may not be changed, i.e. an activity $j, j = 1, , v$, started in mode $m_j, m_j \in \{1, , |M_j|\}$, must be completed in mode m_j without preemption. The objective of the MRCPSP is to find an assignment of modes to activities, as well as precedence- and resource-feasible starting times of all activities, such that project duration is minimized. The MRCPSP is also strongly NP-hard, as a generalization of the RCPSP. Moreover, for more than one non-renewable resource, the problem of finding a feasible solution of the MRCPSP is already NP-complete (Kolisch (1995)).

Using the classification proposed in (Herroelen et al (1999)), the problem is denoted as $m, 1T|cpm, disc, mu|C_{max}$, and $MPS|prec|C_{max}$ according to the notation by Brucker et al (1999).

All the MRCPSP parameters are summarized in Table 6.3.

Table 6.3. Parameters of the MRCPSP.

Symbol	*Definition*
$G(V, E)$	directed AoN graph representing the structure of the project
V	set of activities
$v = \|V\|$	number of activities
$i \longrightarrow j$	precedence constraint between activities i and j
E	set of precedence constraints between activities
$Prec_j$	set of direct predecessors of activity j
$Succ_j$	set of direct successors of activity j
K_R	set of renewable resources
$R = \|K_R\|$	number of renewable resources
R_k	number of units of renewable resource k available in each time period
K_N	set of non-renewable resources
$N = \|K_N$	number of non-renewable resources
N_l	number of units available of non-renewable resource l
M_j	set of execution modes of activity j
m_j	execution mode of activity j
p_{jm}	processing time of activity j executed in mode m_j
r_{jkm}	request for renewable resource k by activity j executed in mode m_j
n_{jlm}	request for non-renewable resource l by activity j executed in mode m_j
$S = \{S_1, S_2, ..., S_v\}$	schedule
S_j	starting time of activity j
C_j	completion time of activity j

6.4 Literature review

Lots of papers concerning setup times have been published throughout a few recent decades. Most of them are devoted to single or parallel machine scheduling problems, as well as to job shop and flow shop problems (see Allahverdi et al (1999) or Yang and Liao (1999) for a survey). Unfortunately, only a few papers have been dedicated to project scheduling problems with setup times. It is mainly caused by the fact that in many cases either setup times are not taken into account explicitly (for example they do not occur in the problem or are relatively small and therefore can be neglected), or it is assumed that setup times are sequence-independent and included in the activity duration.

In the classification presented by Herroelen et al (1999), where notation $\alpha|\beta|\gamma$ is used to describe project scheduling problems, the parameter $\beta_9 \in \{\circ, s_{ij}\}$ denotes setup times: $\beta_9 = \circ$ - no setup times, $\beta_9 = s_{ij}$ - sequence-dependent setup times. In (Demeulemeester and Herroelen (2002)) this classification has been further extended to $\beta_9 \in \{\circ, s_{ij}, \mathrm{s}_{\mathrm{ij}}, \widetilde{s}_{ij}\}$, where $\beta_9 = \mathrm{s}_{\mathrm{ij}}$ denotes stochastic sequence-dependent setup times, and $\beta_9 = \widetilde{s}_{ij}$ denotes fuzzy sequence-dependent setup times. In another classification of project scheduling problems proposed by Brucker et al (1999) setup times are not considered.

Most papers in the area of project scheduling have been dedicated to developing efficient algorithms and generalizing models. Only a few papers describe how to generate problem instances, but most of them consider the classical RCPSP. The paper by Drexl et. al (1999) is the only one where a procedure of generating data for setup times occurring in the MRCPSP has been described. It is assumed that setup times are sequence-dependent and mode-dependent. For each activity $j \in V$, a set W_j is defined, containing those activities for which setup times have to be considered. For every activity $j' \in W_j$, a mode-dependent and sequence-dependent setup time $w_{jmj'm'}$ is defined, which denotes a minimal time that must lapse between the finishing of activity j performed in mode m and the beginning of activity j' performed in mode m'. Moreover, it is assumed that $j \in W_{j'} \Rightarrow j' \in W_j$ and $w_{j'm'jm} = w_{jmj'm'}$, as well as that setup times may be defined for a subset of all possible pairs of project activities, depending on the parameter $CTS \in [0, 1]$. If $CTS = 0$, then no setup times are defined. If $CTS = 1$, then setup times are defined for each pair of activities. The pairs of jobs, for which setup times are defined, are drawn randomly and the length of the corresponding mode-dependent setup time is also drawn randomly from the interval $[MinST, MaxST]$, where $MinST$ and $MaxST$ are input parameters.

The first research on the project scheduling with setups has been made by Kaplan (1991), who considers a single-mode problem with preemptable activities. Whenever a preempted activity is restarted, a setup time is incurred. A dynamic programming procedure is proposed to solve this problem optimally using a unit-time duration model of the RCPSP, where each activity $j \in V$ is split into p_j unit-time-duration activities. Unfortunately, the algorithm based on an incorrect theorem might fail to find the optimal solution, as it has been shown by Demeulemeester (1992).

Kolisch (1995) presents a zero-one programming formulation for the RCPSP with setup times. Two modes representing the processing including and excluding a setup, respectively, are defined for every activity requiring a setup. These two modes differ in the duration of activities only while the resource requests are identical. The renewable resources, which should be appropriately set up in order to process a certain activity, are called setup-resources and the states of these resources after different setups are called setup-states. It is assumed that each setup-resource is available with one unit per period only, and can therefore process only one activity at a time, as well as each activity can use at most one setup-resource. If setup-state u on setup-resource k is required to perform a certain activity j, the resource request r_{jku} is equal to 1 (0 otherwise). A continuous variable y_{kut} is used to check if setup-resource k has setup-state u at the end of period t ($y_{kut} = 1$). Moreover, a setup time heuristic (STH) algorithm based on the parallel schedule generation scheme (SGS), where the concepts of the presented model are used, is proposed to solve the problem.

The general production scheduling problem (GPSP) which is an extension of the job shop scheduling problem is considered by Demeulemeester and Herroelen (1996). The GPSP differs from the job shop scheduling problem mainly in: (i) the existence of more than one machine of a certain type, (ii) the existence of sequence-independent setups when a machine switches from one product to another, and (iii) the existence of precedence constraints between activities, as well as several other features. Thus, the GPSP looks similar to the RCPSPST. It is assumed that batch splitting is not allowed, and a batch has to be completely processed on a machine before it can be moved to the next machine. Under this assumption a sequence-independent setup time can be overlapped with a production time of the same batch on the preceding machine. An AoN network and time lags are used to model the sequence-independent setup times. The nodes (activities) of the network correspond to production batches, while the arcs represent the finish-to-start precedence relations between two activities i and j, where FS_{ij} is not less than $-s_j$.

Dodin and Elimam (1997) consider an audit scheduling problem with the time and cost of the auditor's travel, as he or she changes the assignment from one engagement to another. If an auditor switches from activity i in engagement g to another activity j in engagement g', then this switch may require some travel time and expense, which depend on the location of g' in relation to g or to the home base. It is assumed that the travel occurs during non-working hours and its monetary value is included in the travel cost, that hence is treated as auditor- (resource-) dependent. The travel cost and other expenses incurred during the auditor's change of assignments from one engagement to another are captured by a sequence-dependent setup cost. An ILP model is proposed, where the objective function minimizes the total cost including both tardiness and setup costs as well as the costs of mismatching auditors and audit activities. The precedence constraints are added to this model in order to avoid the TSP formulation of the problem, which can arise when sequence-dependent setup times are considered and lead to a strongly NP-hard problem.

The RCPSP with time windows and sequence-dependent changeover times is considered by Neumann et. al (2003). The set of all renewable resources that must be set up for processing certain activities are called changeover resources. The resource requests by activities and setups may be greater than one, and they must not exceed the respective resource capacities at any time. If this constraint is fulfilled, the resulting schedule is called changeover-feasible. The authors show how to detect whether or not a generated schedule is changeover-feasible.

They also propose a resolving resource conflicts procedure for changeover-infeasible schedules.

Recently, Mika et al (2003) propose a new category of setup times to model times necessary, e.g., to transfer some very huge data files from one computational node to another one in a computational grid. This new category is called schedule-dependent setup times. The problem concerns the scheduling of workflow jobs, which consist of several precedence-related tasks running on certain computational nodes of a grid. Tasks process huge data files and leave them on corresponding nodes or send them to other computational nodes. Transportation times are treated as setup times, and strongly depend on the locations of both nodes: the first one from which the file is transferred, and the second one to which it is uploaded.

6.5 Modelling setup times in the context of project components

As we have already mentioned, the classical unconstrained project scheduling problem can be solved optimally using the CPM, but when resources are available in limited amounts then the resulting problem, known as the RCPSP, is NP-hard. When setup times are involved in the PSP, the problem in most cases can not be solved optimally in polynomial time, even if there are no resource constraints (Baker (1974)). The unconstrained project scheduling problem with setup times is usually denoted as the PSPST, whereas the RCPSP with setup times as the RCPSPST, and the MRCPSP with setup times as the MRCPSPST. Let us remind that in project scheduling with setup times, resources usually can not be treated as sets of identical units to which activities can be assigned arbitrarily. As it was discussed in Section 16.1, all resource units requiring setups should be in most cases considered separately. In Section 16.2 some properties of setups adopted to project scheduling from machine scheduling have been presented, except for the schedule-dependent setups which is a new category of setups arose in the area of project scheduling. Now we will show how some project scheduling components influence setup times. The influence of the precedence constraints, resource availability constraints, multiple resource units requests, multiple resources, multiple modes, and auxiliary resources on setup times will be discussed.

6.5.1 Precedence constraints

While modelling setup times in the presence of precedence constraints, at least 3 different cases can be distinguished: the first one where setups depend on the precedence constraints, the second one where setups do not depend on the precedence constraints, and the last one where setups are partially dependent on precedence constraints.

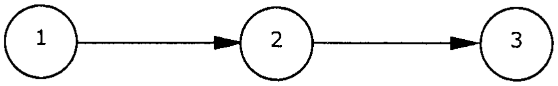


Figure 6.4. A sample project.

In order to explain these three cases we use the following simple example of the RCPSPST. A sample project consists of three activities and two renewable resources R_1 and R_2. The processing times and resource requests are as follows: $p_1 = 2$, $p_2 = 4$, $p_3 = 3$, $r_{11} = 2$, $r_{22} = 2$, $r_{31} = 2$ and $r_{12} = r_{21} = r_{32} = 0$. The resource availabilities are: $R_1 = 2$ and $R_2 = 2$. All setup times are equal to 2. The structure of the project is represented by the AoN network presented in Figure 6.4. It is assumed that all setups are inseparable and sequence-independent.

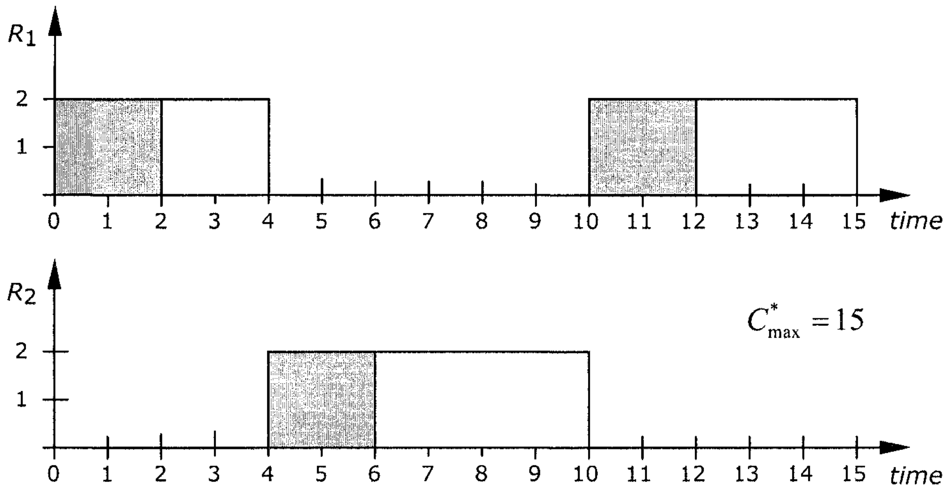


Figure 6.5. A schedule for the sample project with precedence-dependent setups.

In the first case, where setups depend on precedence constraints, no setup associated with activity j can be started before the completion of all direct predecessors of j. For the considered sample project it results in the schedule presented in Figure 6.5, where gray boxes represent setups and white boxes correspond to activities.

Such a situation occurs, e.g., when a resource that does not need any setup has been used by the preceding activity and is required for setting up a resource necessary for the succeeding activity.

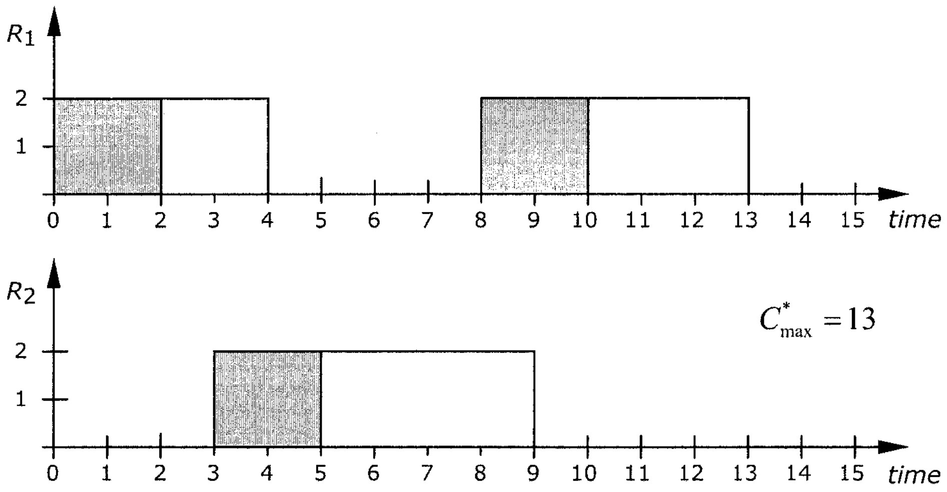


Figure 6.6. A schedule for the sample project with precedence-independent setups.

In the second case, where setups do not depend on precedence constraints, they need not to wait for the completion of the preceding activities. These setups can be executed in parallel with the preceding activities. In this situation the resulting schedule for the sample project is presented in Figure 6.6.

Such a situation occurs, e.g., when setups require only resources necessary for their execution and do not require any auxiliary resource (see section 6.5.6). In the last case setups partially depend on precedence constraints. It means that a determined part of a setup can be executed before the completion of the preceding activities. Let us assume that in the considered example a part of each setup equal to one time period can be executed before the completion of the preceding activity. Then the resulting schedule is presented in Figure 6.7.

Such a situation can occur, e.g., when some resources are modified during the execution of the preceding activity, and then they must be transported or transferred to another place where they are necessary for setting up some resources required by the succeeding activity. A setup may start as early as the first set of resources is ready to be transported.

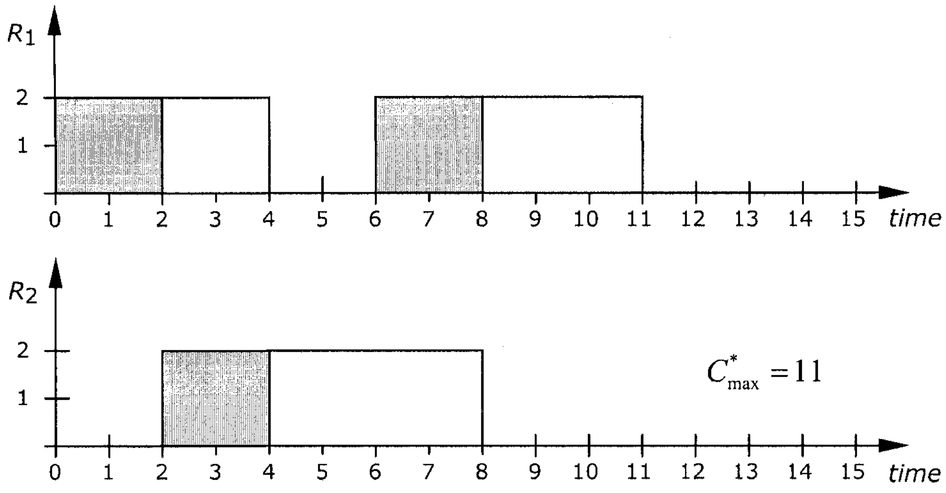


Figure 6.7. A schedule for the sample project with partially precedence-dependent setups.

6.5.2 Resource constraints

In the classical unconstrained project scheduling problem (PSP) it is assumed that all renewable resources are available in amounts (numbers of units) sufficient to perform all activities in parallel. In consequence, the schedule is build taking into account the precedence constraints only. However, under resources constraints the resulting RCPSP problem becomes much more complex.

Let us consider another sample project (with $v = 6$ activities and one renewable resource), represented by the AoN graph shown in Figure 6.8. The durations and resource requests of all activities are presented in Table 6.4.

Table 6.4. Parameters of the sample project no. 2.

Activity - j	*Duration - p_j*	*Resource request - r_{jl}*
1	2	1
2	1	2
3	4	2
4	3	1
5	4	1
6	1	3

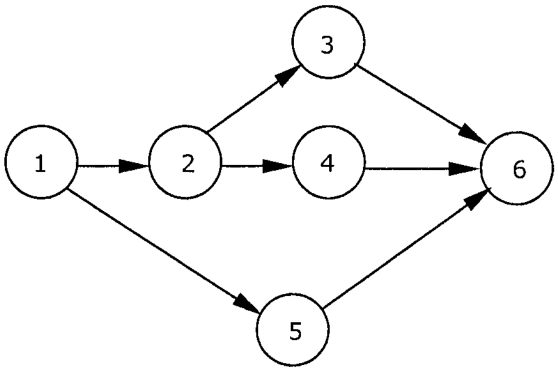


Figure 6.8. A sample project no. 2.

The schedule obtained using the CPM is presented in Figure 6.9. One can notice that if resource R_1 is available in at least four units, the considered problem is a resource-unconstrained project scheduling problem.

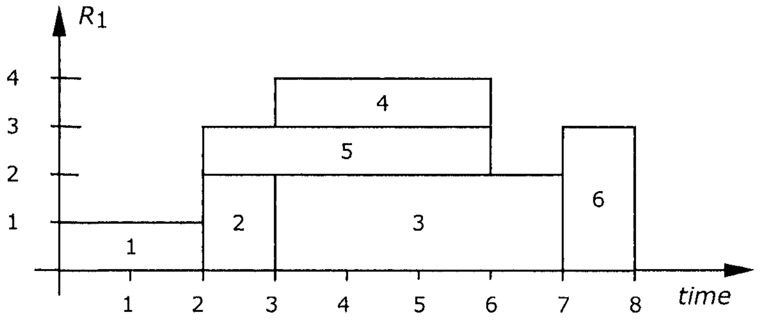


Figure 6.9. A schedule for unconstrained project no. 2.

Now, let us assume that each activity requires a setup time. Moreover, let us also assume that each setup time is separable, sequence-independent, precedence-independent, and is equal to one time unit. In the next two figures schedules obtained for the sample project no. 2 with setups assuming that $R_1 = 4$ (Figure 6.10) and $R_1 = 8$ (Figure 6.11) are presented.

It is easy to observe that increasing the number of available units of a renewable resource results in shortening the project duration. This observation leads to the next conclusion which can be formulated as follows. There are possible

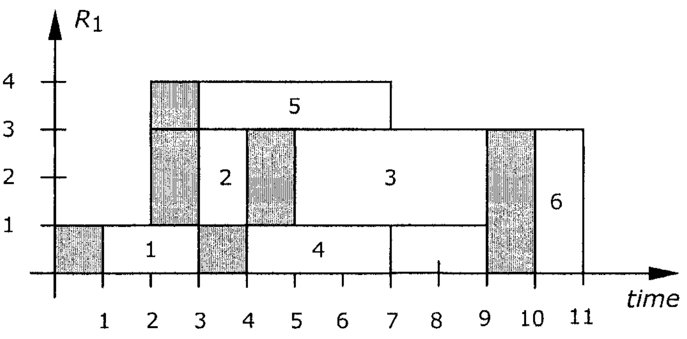


Figure 6.10. A schedule for project no. 2 with setups and $R_1 = 4$.

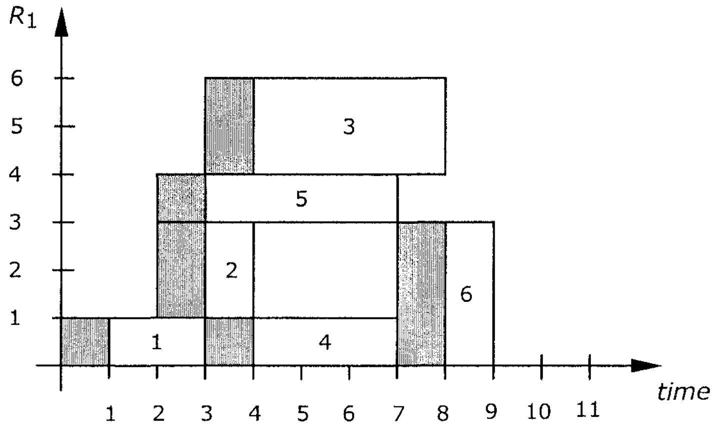


Figure 6.11. A schedule for project no. 2 with setups and $R_1 = 8$.

situations where projects are not resource-constrained, but in the presence of setups the resource availabilities can not be neglected because they may restrict the possibilities of execution some activities and setups in parallel.

Let us examine the schedule presented in Figure 6.10. The setup associated with activity 3 could be executed in the 4-th time period together with activities 2 and 5, as well as with the setup for activity 4, but - unfortunately - in this time period there are not enough idle resource units available which could be used to perform the setup for activity 3. Similar situation occurs for the setup associated with activity 6. So, this is a case where resources are not constrained with respect to the possibilities of the execution of all activities, but they are constrained for the execution of some setups. We call such constraints *soft resource constraints*, in opposition to the classical resource constraints which we call *hard resource constraints*.

Let $r_k^{max} = \max_{t=1,\ldots,C^*_{max}} \left\{ \sum_{j=1}^{v} r_{jk} \cdot x_{jt} \right\}$, where x_{jt} is equal to 1 if activity j is executed at time period t and 0 otherwise, and C^*_{max} is the project duration calculated using the CPM for the problem with neglected setups and resource constraints. In other words r_k^{max} denotes the maximal total temporal request for renewable resource k assuming that the schedule is build according to the CPM. If $R_k < r_k^{max}$ then the constraints on resource k are hard, if $r_k^{max} \leq \delta$ then they are soft. $\delta \leq 2 \cdot r_k^{max}$ is a value that denotes the maximal temporal request for resource k that depends on the structure of the project with regard to setups. Finally, if $R_k \geq \delta$ then resource k is unconstrained.

6.5.3 Multiple resource units

The next difference between machine scheduling and project scheduling is the number of resource units. In machine scheduling each machine is treated separately as a single unit resource, even if these are identical parallel machines. In project scheduling such a situation occurs rather seldom. More frequently resources contain more than one unit, and all units are treated equally. In this section the influence of the multiplicity of resource units on setup times is discussed.

6.5.3.1 Single-unit renewable resource. Single-unit renewable resources occur rather seldom in project scheduling but, of course, their existence is possible in practice, especially when there are no other resources and the constraints on those resources are hard. An example of such a resource may be a skilled worker or an expert, who can not be substituted by other workers. If a setup is necessary to prepare a single unit resource for the execution of a given activity, then it is known in advance on which resource unit the activity will be executed. Thus, it is also known in advance which resource unit requires the setup. This property allows to use models and methods adapted from single machine scheduling with setup times.

6.5.3.2 Renewable resource with multiple units. When the number of units of a given renewable resource is greater than one then the problem becomes more complex. In such a problem two cases can be distinguished: the first one where resource requests of all activities are equal to one, and the second one where there are also activities with resource requests greater than one.

In the first case, some models and methods developed for parallel machine scheduling problems with setup times can be easily adapted to project scheduling problems with single constrained resource (with $R_1 > 1$) and unary resource requests.

In the second case, more than one requested resource unit have to be set up in order to execute some activities on these resource units. First of all, the decision on which units the setup will be prepared has to be made. It is a very important decision because the corresponding activity has to be executed on exactly the same resource units. Now, there are two further possibilities: 1) an *undivided setup,* where all requested resource units should be set up at the same time (Figure 6.12), 2) a *divided setup,* where each requested resource unit can be set up at an arbitrary time before starting the associated activity (Figure 6.13). The first situation is typical for inseparable setups but is also possible for separable ones, while the second one occurs only for problems with separable setups and, for example, with strongly constrained auxiliary resources (see section 6.5.6) necessary to set up the requested resources.

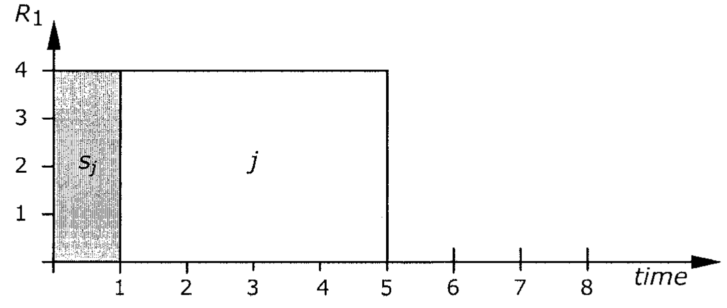


Figure 6.12. Undivided setup for a multiunit resource.

Moreover, when sequence-dependent or schedule-dependent setup times are considered together with divided setups, it is possible that setup times for particular requested units of the very same resource will differ among themselves. For example, if at least two auditors are necessary in one place to execute a given activity, then they must travel from the current location to the place where the activity will be executed. Thus, the setup times for particular units may differ and in this case depend on the current location of these units (auditors).

6.5.4 Multiple resources

In many project scheduling problems there are more than one resource available. Activities may require for their processing one resource only, and then the

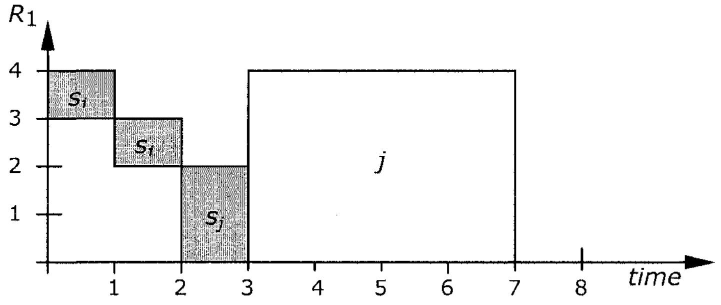


Figure 6.13. Divided setup for a multiunit resource.

problem of scheduling activities and setups of the project reduces to a problem similar to that described in the previous section. But, of course, it is also possible that activities request for more than one resource, and setups are required for all these resources. In such a case, each setup s_j, necessary to prepare all the required resources for the execution of activity j, consists of a set of setups σ_j^k, where σ_j^k is the setup concerning particular resource k. It is obvious that different resources require different setups which, of course, may take different amounts of time. This observation leads us to the next categorization of setups, where *synchronous, semi-synchronous,* and *asynchronous* setups are distinguished. In order to explain these terms, it is assumed that a given activity requests for several resources and each requested resource requires a setup. The time needed to set up is different for each resource.

In the case of the *synchronous setup* (Figure 6.14), all setups σ_j^k necessary to set up all requested resources for the execution of activity j have to be performed in parallel. The time needed to perform the entire setup s_j is equal to the execution time of the longest setup σ_j^k. Of course, the duration of the parallel execution of setups σ_j^k is at most equal to the execution time of the shortest setup from among all of them. Moreover, each resource required by a setup is exclusively allotted to this setup over the whole setup time, and is not available for other setups or activities during this time interval. In other words, the entire synchronous setup must be performed as shortly as possible, and all resources used for it may not be assigned to other activities or setups at the same time.

The *semi-synchronous setup* (Figure 6.15) is very similar to the synchronous one. The main difference is that the resources are not exclusively allotted to a setup. They can be used by other setups within the setup time, because they are not blocked for this time interval.

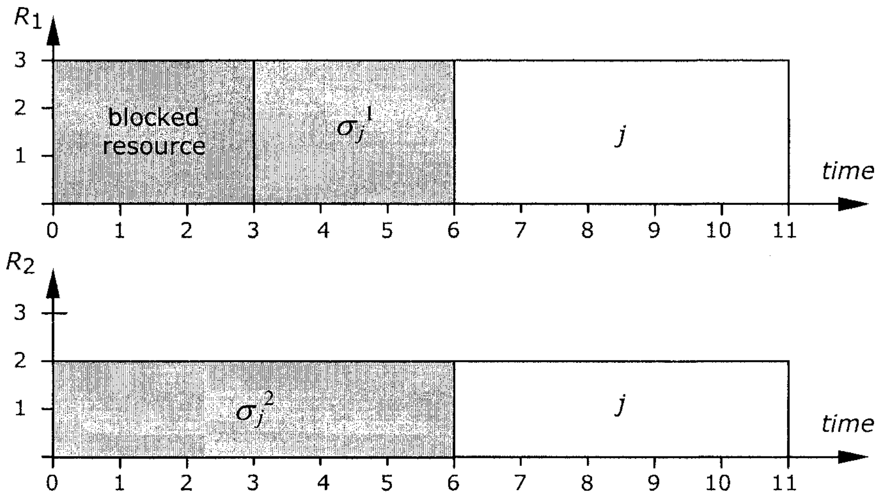


Figure 6.14. An example of the synchronous setup.

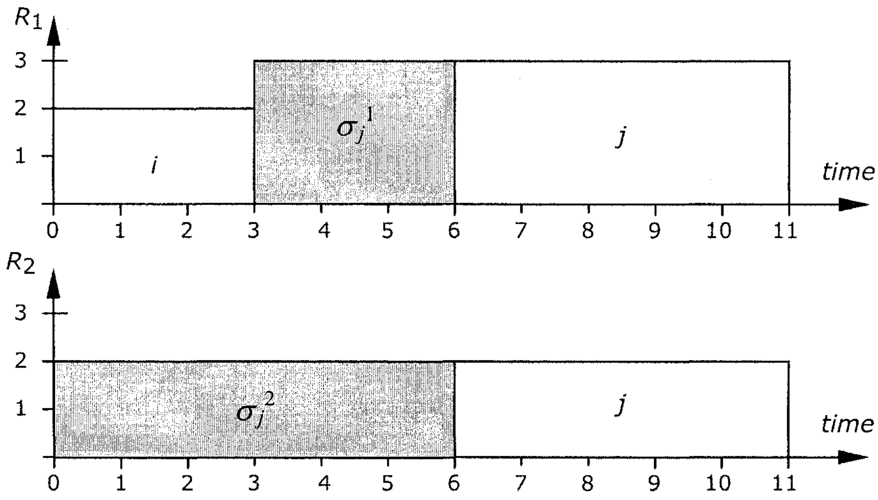


Figure 6.15. An example of the synchronous setup.

An example of a situation in which semi-synchronous setup occurs is making a rib-and-slab floor of a building, where an activity is to place a concrete on a previously prepared floor of the building. There are at least two constrained resources necessary to complete this activity: a concrete pump and a truck concrete mixer. Both machines need a setup which consists in moving them from the concrete-mixing plant to the construction site. Both trucks should arrive at the destination almost at the same time. Otherwise, either the concrete may become useless if the concrete pump comes too late, or we have to pay additional money for the working hours of the concrete pump waiting for the truck concrete mixer.

In the case of the *asynchronous setup* (Figure 6.16) each particular setup σ_j^k can be performed independently of the other ones comprising the same setup s_j. In other words, they do not need to be performed simultaneously.

As an example of a situation where asynchronous setup may occur, a machining of a detail made of some metal may be used. These activity usually requires setting up at least two constrained resources: a machine tool and a cutting tool. The detail must be mounted in a proper chuck of the machine tool, whereas the cutting tool must be mounted in a given fixture. Both setups may be performed sequentially by one person.

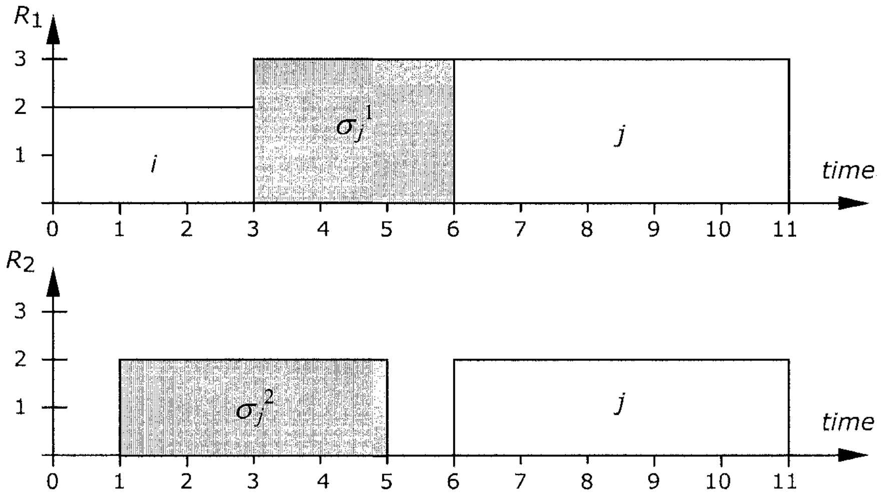


Figure 6.16. An example of the asynchronous setup.

6.5.5 Multiple modes

All previously discussed models of setups can also be considered in the multi-mode resource-constrained project scheduling problem with setup times (MRCPSPST). Moreover, each execution mode for an activity of the MRCPSP usually requires its own combination of resources, which is different from the resource combinations for other execution modes. Thus, it is obvious that different setups have to be associated with different modes. We call them *setup modes.*

Moreover, it is possible that there are more than one alternative ways of performing a setup for a given execution mode of an activity. Such a situation occurs usually when auxiliary resources are used for a setup. As a result, the number of setup modes can be greater than the number of execution modes.

On the other hand, it is also possible that the number of feasible execution modes can be reduced because of the feasibility of setup modes. Such a case appears when constrained auxiliary resources are necessary to set up some resources.

6.5.6 Auxiliary resources

In some situations, in order to perform a setup on a given resource, one or more auxiliary resources are needed. These auxiliary resources are used only for setting up other resources, and are not used during the execution of any project activity. An example of such an auxiliary resource is a group of skilled workers who are capable of and responsible for proper setting up specialized machines. These additional resources can be neglected if they are available in numbers of units sufficient to perform all possible setups in parallel. Otherwise they must be explicitly taken into account as one of the parameters of the problem.

Moreover, if the number of auxiliary resources is greater than one, it is possible in some cases to perform a setup in one from two or more alternative ways, which may differ between themselves in the duration of the setup and resources necessary to perform it. In other words, setups can be executed in one from multiple setup modes.

As an example illustrating the setup modes we use workers (a typical example of auxiliary resources) who are necessary to set up machines, computers, robots, and other resources. Usually workers can be divided into several groups according to their skills. Let us assume that there are at least three groups of workers: usual workers, skilled workers, and experts. Workers from all the three groups are able to set up some resources for executing some activities. For example, a certain machine may be set up for executing a given activity by two experts, one expert and two skilled workers, one expert and four usual workers, or by four skilled workers. In consequence, four different setup modes have been obtained.

6.6 Possible extensions of the presented models

In the previous section the influence of project components on setup times has been discusses. On this basis a few models of setup times have been described, including introducing some new categories of setups, like: precedence-independent, precedence-dependent, and partially precedence-dependent setups, divided and undivided setups, synchronous, asynchronous, and semi-synchronous setups, and also multiple setup modes. However, in this paper we only consider a certain class of project scheduling problems, and the goal of this section is to show possible extensions of the presented models, as well as other directions of further research in this area.

Let us first recall that we consider static, deterministic, and discrete project scheduling. It means that similar analyses can be performed for dynamic project scheduling (on-line scheduling), stochastic project scheduling, where the parameters of the problem may have stochastic settings, or continuous project scheduling, where time is not discretized and has continuous nature (there is no scheduling period). However, since static, deterministic, and discrete project scheduling is the subject of most research, we will now focus on possible extensions of this class.

Let us start with the optimization criterion. We have only considered the minimization of the makespan (i.e. project duration) as the scheduling criterion. It is obvious, that other criteria can be considered as well, both regular (e.g. minimizing the mean flow time, the project lateness or tardiness, etc.), and non-regular (e.g. minimizing the weighted earliness-tardiness of the project). From among the non-regular criteria, the financial ones seem to have strong practical justification in the context of setup costs, for example the maximization of the net present value (NPV). But also time/cost trade-off problems with setup costs have not been extensively studied yet, to the best of our knowledge. Many other criteria can be given too, as well as multi-objective project scheduling with setup times (costs) which could be a big challenge for the researchers.

Next, preemption may be allowed on activities and/or setups which leads us to the preemptive resource-constrained project scheduling problem with setup times. Another extension can be imposing ready times on activities, and/or duedates on activities or the entire project. Ready times and duedates (or even deadlines) may be imposed on setup operations as well. Moreover, other types of precedence constrains than finish-to-start with no time lags could be analyzed, which leads us to the resource-constrained project scheduling problem with generalized precedence constraints and setup times.

Besides, other resource categories may be taken into account, e.g. non-renewable resources or partially renewable resources, and their influence on setup times. Also, constant or variable resource requests can be considered, as well as continuous resources where the processing rate (or processing time) of an activity depends on the amounts of the continuous resources allotted to this activity at a time. The time/resource trade-off problem with setup times could be another case to consider.

All the above extensions may be also studied in the context of multiple modes. Especially, the so-called mode identity constrains should be analyzed in combination with class setups. The case of mode identity constraints is a generalization of the multi-mode case where the set of all activities is partitioned into disjoint subsets, while all activities forming one subset have to be processed in the same mode. Time and cost incurred by processing a subset of activities depend on the resources assigned to it. The connection to the case of class setup is obvious, but the resulting problem needs further attention.

Finally, project scheduling with setup times can be also approached from the sensitivity and robustness point of view, which is a direction in project scheduling intensively studied in recent years because of great practical importance. Also other aspects, concepts, and models in project scheduling, which we have not been able to mention here, may be used as possible extensions. Certainly, project scheduling problems with setup times (costs) require extensive further research because of a huge number of practically justified cases.

6.7 Conclusions

In this chapter project scheduling problems with setup times have been considered. An extensive classification of setups in project scheduling has been presented. The definitions of a new category of setups - schedule-dependent setups, as well as precedence-independent, precedence-dependent, and partially precedence-dependent setups, divided and undivided setups, synchronous, asynchronous, and semi-synchronous setups, and also multiple setup modes have been given. Modelling setup times in the context of various project scheduling components has been widely discussed. Some possible extensions of the presented models have been pointed out.

Let us finally stress that we have attempted to consider the title issue as comprehensively as possible. However, we are aware of the restrictions of this research, as well as of the complexity of the undertaken subject, in general. We have tried to propose possible extensions not considered in this paper, but even so, we could not mention all the cases which might appear in practice. Certainly, project scheduling problems with setup times require further attention, because of their interesting theoretical properties, as well as of a great variety of practical applications.

6.7.1 Acknowledgements

This research has been supported by the Ministry of Science and Education, grant no 3T11F 025 28.

We express our gratitude to Rainer Kolisch from the Technical University of Munich and Christoph Schwindt from the Clausthal University of Technology, Germany, for their valuable help.

References

Allahverdi, A., Gupta, J. N. D., and Aldowiasan, T. (1999). A review of scheduling research involving setup considerations, *Omega, International Journal of Management Science* 27:219–239.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling,* John Wiley & Sons, New York.

Błażewicz, J., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1983). Scheduling subject to resource constraints, *Discrete Appl. Math.*5:11–24.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* 112:3–41.

Bruno, J., and Downey, P. (1978). Complexity of task sequencing with deadlines, set-up times and changeover costs, *SIAM Journal on Computing* 7:393–404.

Chen, B. (1993). A better heuristic for preemptive parallel machine scheduling with batch setup times, *SIAM Journal on Computing* 22:1303–1318.

Chrétienne, P., and Picouleau C. (1995). Scheduling with communication delays: A survey, in: *Scheduling Theory and its Applications,* P. Chrétienne, E.G. Coffman, Jr., J.K. Lenstra, and Z. Liu, eds., John Wiley & Sons, New York, pp. 65–90.

Conway, R.W., Maxwell, W.L., and Miller, L.W. (1967). *Theory of Scheduling,* Addison-Wesley, Reading, Massachusetts.

Demeulemeester, E. (1992). Optimal Algorithms for Various Classes of Multiple Resource-Constrained Project Scheduling Problems, Dissertation, Katholieke Universiteit Leuven, Belgium, (unpublished).

Demeulemeester, E.L., and Herroelen, W.S. (1996). Modelling setup times, process batches and transfer batches using activity network logic, *European Journal of Operational Research* 89:355–365.

Demeulemeester, E.L., and Herroelen, W.S. (2002). *Project Scheduling: A Research Handbook,* Kluwer Academic Publishers, Norwell.

Dodin, B, and Elimam, A.A. (1997). Audit scheduling with overlapping activities and sequence-dependent setup costs, *European Journal of Operational Research* 97:22–33.

Drexl, A., Nissen, R., Patterson, J.H., and Salewski, F. (2000). ProGen/πx - An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions, *European Journal of Operational Research* 125:59–72.

Herroelen, W., Demeulemeester, E., and De Reyck, B. (1999). A classification scheme for project scheduling, in: *Project Scheduling: Recent Models, Algorithms and Applications,* J. Węglarz, ed., Kluwer Academic Publishers, Norwell, pp. 1–26.

Kaplan, L. (1991). Resource-constrained Project Scheduling With Setup Times, Working Paper, Department of Management Science, University of Tennessee, Knoxville, USA, (unpublished).

Kelley, J.E., Jr. (1961). Critical-path planning and scheduling: Mathematical basis, *Operations Research* 9:296–320.

Kolisch, R. (1995). *Project Scheduling under Resource Constraints - Efficient Heuristics for Several Problem Classes,* Physica, Heidelberg.

Mika, M., Waligóra, G., and Węglarz, J., (2003). A metaheuristic approach to scheduling workflow jobs on a grid, in: *Grid Resource Management: State of the Art and Future Trends,* J.M. Schopf, J. Nabrzyski, and J. Węglarz, eds., Kluwer Academic Publishers, Norwell, pp. 295–318.

Monma, C.L., and Potts, C.N. (1989). On the complexity of scheduling with batch setups, *Operations Research* 37:798–804.

Neumann, K., Schwindt, Ch., and Zimmermann, J. (2003). *Project Scheduling with Time Windows and Scarce Resources : Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions,* 2nd ed., Springer, Berlin.

Schwindt, Ch. (2005). *Resource Allocation in Project Management*, Springer, Berlin.

Yang, W.H., and Liao, Ch.J. (1999). Survey of scheduling research involving setup times, *International Journal of Systems Science* 30:143–155.

II

ALGORITHMS

Chapter 7

LOWER BOUNDS FOR RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

Recent advances

Emmanuel Néron
LI, Université François-Rabelais de Tours
Polytech'Tours, 64 av. Jean Portalis 37200 Tours France
emmanuel.neron@univ-tours.fr

Christian Artigues
LIA - CERI
Université d'Avignon et des Pays de Vaucluse
339 chemin des Meinajariès, BP1228 84911 Avignon Cedex 9 France
Centre de recherche sur les transports Université de Montréal
C.P. 6128, succursale Centre-ville Montréal, QC H3C 3J7 CANADA
christian.artigues@lia.univ-avignon.fr

Philippe Baptiste
CNRS LIX,
Ecole Polytechnique
91128 Palaiseau, France
Philippe.Baptiste@polytechnique.fr

Jacques Carlier
CNRS HeuDiaSyC,
Université de Technologie de Compiègne
Centre de recherches de Royallieu
60200 Compiègne, France
jacques.carlier@utc.fr

Jean Damay
CNRS LIMOS,
Université Blaise Pascal
Complexe scientifique des Céseaux 63177 Aubière Cedex, France.
damay@isima.fr

Sophie Demassey
Centre de Recherche sur les Transports,
Université de Montréal
C.P. 6128, succursale Centre-ville Montréal, QC H3C 3J7, Canada
sophie.demassey@lia.univ-avignon.fr

Philippe Laborie
Ilog France
9, rue de Verdun BP 85,
94253 Gentilly Cedex, France
laborie@ilog.fr

Abstract We review the most recent lower bounds for the makespan minimization variant of the Resource Constrained Project Scheduling Problem. Lower bounds are either based on straight relaxations of the problems (e.g., single machine, parallel machine relaxations) or on constraint programming and/or linear programming formulations of the problem.

Keywords: Resource Constrained Project Scheduling Problem, lower bounds, constraint programming, linear programming.

7.1 Introduction: bounding RCPSP

In the Resource Constrained Project Scheduling Problem (RCPSP), non-preemptive activities requiring renewable resources subject to precedence and resource constraints have to be scheduled to minimize makespan. Our aim is to review recent advances in lower bounding techniques for this fundamental problem.

For a description of exact or heuristic resolution methods, we refer to the recent general state-of-the art surveys on the RCPSP, e.g., (Brucker et al (1999)) and (Demeulemeester and Herroelen, 2002).

Most of the bounds described in this paper are "destructive" bounds (Klein and Scholl, 1999). The mechanism of such bounds is rather simple: a trial value D is fixed for the makespan. If we are able to prove that the corresponding problem has no feasible solution, then $D + 1$ is a valid lower bound. A binary search is performed on the trial value D to find the largest value for which we can prove that no feasible solution exists. Of course the key component of a destructive bound relies on the way we can prove that there is no solution.

This chapter is organized as follows:

- Section 7.2 is dedicated to classical relaxations of the RCPSP to single resource problems, they are based on the single machine problem, the identical parallel machine problem and the Cumulative Scheduling Problem. Bounds presented in this section may not be the most efficient ones but they are fast to compute.

- Section 7.3 presents constraint propagation techniques used to strengthen the problem. These techniques are useful especially if they are used as preprocessing for more complex lower bounds such as the ones based on some linear programming formulation.

- In Section 7.4 we present more sophisticated lower bounds that take into account simultaneously the precedence constraints and several resources. Most of these bounds are efficient but highly time consuming.

We do not develop in this chapter the description of the classical lower bounds based on longest paths computations in the project network and their extensions.

We refer to (Demeulemeester and Herroelen, 2002) for a complete description of these bounds.

Notation used in this chapter are recalled in Table 7.1.

Table 7.1. Notation

Notation	Definition
Activity related data	
$\mathcal{A}$	set of activities
n	number of activities: $n = \|\mathcal{A}\|$
$G(\mathcal{A}, E)$	activity-on-node graph used to model classical precedence relationship
p_i	processing time of activity i
Resource related Data	
$\mathcal{R}$	set of renewable resources
m	number of resources: $\|\mathcal{R}\|$
$r_{i,k}$	number of units of resource k required by activity i
R_k	capacity of resource k
Other Notation	
ES_i	earliest starting time of activity i (release date)
LF_i	latest finishing time of activity i (deadline)
EF_i	earliest finishing time of activity i ($ES_i + p_i$)
LS_i	latest starting time of activity i ($LF_i - p_i$)
S_i	starting time of activity i
C_i	completion time of activity i
D	trial value used to compute destructive lower bound
q_i	tail of activity i: $q_i = D - LF_i$

7.2 Single resource problems

In this section, we present classical methods for bounding single resource problems that arise as relaxations of the RCPSP. Basically these new problems are obtained either by relaxing the precedence relations to time-windows for activities and/or by relaxing partially the resource constraints. Namely these problems are (1) the single machine problem, (2) the identical parallel machine problem and (3) the Cumulative Scheduling Problem (CuSP). All these problems involve release dates and tails (or deadlines). For these three relaxations, we first describe how they can be get from an initial RCPSP instance, and then we describe lower bounds and time-bound adjustments methods that can be used in turn to get lower bounds for the initial RCPSP.

7.2.1 Disjunctive based bounds

In the single machine problem, a set of activities has to be processed on a single machine. Each activity,i has a release date ES_i, a processing time p_i, a tail (q_i) or a deadline LF_i. This problem is known to be $\mathcal{NP}$-Hard in the strong sense. It plays a central role for solving the RCPSP as (1) we can derive release dates and tails of activities from the precedence constraints and (2) we can compute sets of activities that, due to machine or to precedence constraints, cannot be processed in parallel and can thus be "put"on a fictive single machine.

In this part we present both a method to build such single machine problems from an initial RCPSP and classical techniques either to compute lower bounds of the makespan ($\max_i(C_i + q_i)$, where C_i denotes the completion time of the activity i) or to derive satisfiability tests, and time-windows adjustments of activities.

7.2.1.1 Maximum clique computation. Consider the decision variant of the RCPSP. The aim of this section is to build a set of single machines (a machine is a resource with unit capacity), on which some activities have to be processed.

Such redundant machines are useful: on each redundant single machine, edge-finding constraint propagation can be applied (see Section 7.2.1.3) and thus, the time-windows of activities can be tightened.

To generate redundant single machines, we look for sets of activities that are known not to overlap in any feasible solution. Note that two activities i and j never overlap in time (1) if there is a precedence constraint between i and j or (2) if there is a resource such that the total amount of capacity required by i and j on the considered resource exceeds its capacity. In the following, two activities meeting conditions (1) and/or (2) are said to be "compatible". Any set of activities in which all activities are pairwise compatible is a candidate redundant machine.

We associate a binary variable $\Upsilon_i \in \{0, 1\}$ with each activity i (Υ_i equals 1 when i belongs to the single machine under construction, 0 otherwise). A vector Υ corresponds to a valid redundant machine if for all activities i, j that are not compatible, $\Upsilon_i + \Upsilon_j \leq 1$.

Since the edge-finding constraint propagation algorithm is costly in terms of CPU time, very few redundant machines can be generated. Hence, we have to heuristically select some of them. Our intuition is that "good" redundant machines are heavily loaded. So, we try to find a vector Υ that maximizes $\sum p_i \cdot \Upsilon_i$. The resulting problem is a MIP with n variables and at most n^2 constraints (much less in practice). In (Baptiste and Le Pape, 2000), a greedy heuristic is used to build a solution to a similar MIP. Initial experiments have shown that, in terms of final reduction of time-windows, it is much better to solve the MIP to optimality.

More precisely, we build one global redundant machine according to the above MIP and one redundant machine per initial RCPSP resource. For each cumulative resource, a redundant machine is created in which all the activities requiring more than half of the resource are put (such activities never overlap in time). We then try to add some extra activities on this redundant machine. To do so, the MIP is modified by replacing variable corresponding to activity h, i.e., Υ_h the variable corresponding to activity that is already put on the redundant machine by is replaced by 1. Then a "reduced" MIP, that is solved to optimality, is get.

7.2.1.2 One machine problem lower bound. Several lower bounds have been proposed for such one machine problem, i.e., with release dates and tails or deadlines. The commonly used lower bound of the C_{max} for the single machine problem is

$$max_{(h,l)\in\mathcal{A}^2} ES_h + q_l + \sum_{i:ES_i\geq ES_h, q_i\geq q_l} p_i.$$

This maximum can be computed in $O(n \log n)$ steps thanks to Jackson Preemptive Schedule, the preemptive schedule associated with the Largest Tail First dispatching rule.

7.2.1.3 Edge-finding and time-bound adjustments. Now, we consider the decision variant of the single machine problem as defined earlier. So, we have time-windows $[ES_i, LF_i]$ (release date / deadline) in which activities have to be processed.

Edge-Finding and time-bound adjustments (Carlier and Pinson, 1989, Applegate and Cook, 1991) consist of deducing that some activities from a given set Ω must, can, or cannot, be executed first (or last) in Ω. Such deductions lead to new ordering relations ("edges" in the graph representing the possible orderings of activities) and new time bounds, i.e., strengthened release dates and deadlines. The edge-finding algorithm is one of the most well known OR algorithm integrated in CP (Baptiste et al., 1999). It is a very efficient global constraint propagation algorithm for disjunctive non-preemptive scheduling.

In the following, ES_Ω denotes the smallest release date among the activities in Ω. Similarly, LF_Ω is the largest deadline among activities in Ω. Finally, let p_Ω be the sum of the processing times of the activities in Ω. Let $i \ll j$ $(i \gg j)$ means that i is executed before (after) j and $i \ll \Omega$ $(i \gg \Omega)$ means that i is executed before (after) all the activities in Ω. Once again, variants exist (see Baptiste et al., 1999 for a review) but the following rules capture the "essence" of the Edge-Finding bounding technique:

$$\forall\Omega, \forall i \notin \Omega, [LF_{\Omega\cup\{i\}} - ES_\Omega < p_\Omega + p_i] \Rightarrow [i \ll \Omega]$$
$$\forall\Omega, \forall i \notin \Omega, [LF_\Omega - ES_{\Omega\cup\{i\}} < p_\Omega + p_i] \Rightarrow [i \gg \Omega]$$
$$\forall\Omega, \forall i \notin \Omega, [i \ll \Omega] \Rightarrow [LF_i = \min(LF_i, \min_{\emptyset\neq\Omega'\subseteq\Omega}(LF_{\Omega'} - p_{\Omega'}))]$$
$$\forall\Omega, \forall i \notin \Omega, [i \gg \Omega] \Rightarrow [ES_i = \max(ES_i, \max_{\emptyset\neq\Omega'\subseteq\Omega}(ES_{\Omega'} + p_{\Omega'}))]$$

If n activities require the resource, there are a priori $O(n \cdot 2^n)$ pairs (i, Ω) to consider. An algorithm that performs all the time-bound adjustments in $O(n^2)$ is presented in (Carlier and Pinson, 1990). Another variant of the Edge-Finding technique is presented in (Carlier and Pinson, 1994). It runs in $O(n \log n)$ but requires much more complex data structures.

7.2.2 Cumulative problem based lower bounds

One simple way to extend single machine based lower bounds for the RCPSP is to consider that more than one activity can be in process at a time point. Then several lower bounds based on either identical parallel machine problem or Cumulative Scheduling Problem have been proposed in the literature. Some of these bounds are presented in this section.

7.2.2.1 Deducing identical parallel machine problems. In the identical parallel machine problem, a set $\mathcal{A}$ of activities has to be processed on π identical parallel machines. Each activity, $i \in \mathcal{A}$, has a release date ES_i, a processing time p_i and a tail q_i or a deadline LF_i. These problems are known to be $\mathcal{NP}$-Hard in the strong sense. Preemptive or semi-preemptive (see Section 7.2.2.2) relaxations can be solved efficiently and then used to compute lower bounds for the RCPSP. Thus building relevant identical parallel machine problem, considering that release dates and tails of activities are deduced from the precedence constraints of the initial RCPSP, is useful for bounding RCPSP.

To build such π-machine problems, let us consider a resource k and a set $\mathcal{J} \subseteq \mathcal{A}$ of activities of the initial RCPSP such that $|\mathcal{J}| = \pi+1$ and $\sum_{i\in\mathcal{J}} r_{i,k} > R_k$, then at most π activities of $\mathcal{J}$ can be in process at the same time. Thus $\mathcal{J}$ defines a π-machine problem, and the optimal makespan of this identical parallel machine problem is a lower bound of the makespan of the initial RCPSP.

In (Carlier and Latapie, 1991) π-machine problems are built such that:

- any activity satisfying $\left\lceil \frac{(\pi+1)r_{i,k}}{R_k} \right\rceil - 1 \geq \eta_i$, ($\eta_i \in \mathbb{N}$), is replaced by η_i identical activities in $\mathcal{J}$,
- activities such that $\sum_{i=1}^{\pi} r_{[i],k} > R_k$, (where $r_{[i],k}$ the $i-th$ smallest resource requirement of the activities of $\mathcal{J}$) are added to $\mathcal{J}$.

7.2.2.2 Bounding identical parallel machine problem.

Here we consider lower bounds and satisfiability tests for the decision variant of the identical parallel machine problem with release dates and tails or deadlines. Satisfiability tests are used to prove that there exists no solution in which activities are processed within their time-windows $[ES_i, LF_i]$, without violating the machine constraint. Then they can be used to compute destructive lower bounds for the initial RCPSP instance.

Preemptive and semi-preemptive relaxations for identical parallel machine problem. In this section, we focus on two satisfiability tests for identical parallel machine decision problem: the first one is based on the preemptive relaxation, that can be solved in polynomial time, the second one uses the notion of mandatory parts of activities into the classical max-flow formulation. These methods are presented through an example.

Table 7.2. Instance of identical parallel machine problem

$i \in \mathcal{A}, \pi = 2$	1	2	3
ES_i	23	0	0
p_i	14	45	74
LF_i	37	46	89

It is well known that preemptive relaxation of the identical parallel machine problem is polynomially solvable and that an optimal solution can be found using a max-flow formulation (Horn, 1974): there exists a flow equal to $\sum_{i \in \mathcal{A}} p_i$, in graph G if and only if there exists a preemptive feasible schedule. We refer to (Brucker, 2002) for a formal description of the graph construction.

The figure 7.1 presents the graph associated with the instance presented in figure 7.2. The computation of this satisfiability test can be done in $O(n^3)$ time.

In a recent paper (Haouari and Gharbi, 2003) propose to build *semi-preemptive schedule*. It is based on a technical result to enforce some parts of an activity to be processed into a given time-interval. They proved that, if an activity $i \in \mathcal{A}$ verifies $LF_i - ES_i < 2 \cdot p_i$, then one machine processes activity i during $[LF_i - p_i, ES_i + p_i]$. This constraint can be simply taken into account by adding minimum flow constraints on arcs between those activities and corresponding time-intervals, once relevant time-points $(LF_i - p_i, ES_i + p_i, \forall i \in \mathcal{A}$ s.t. $LF_i - ES_i < 2 \cdot p_i)$ have been added. Such a graph is presented below. If it does not exist a flow equal to $\sum_{i \in \mathcal{A}} p_i$ in graph G, then there does not exist a non-preemptive feasible schedule. The computation of this satisfiability test is still in $O(n^3)$ time, due to the fact that the number of nodes is still in $O(n)$. Such a graph corresponding to our previous example is presented in Figure 7.2.

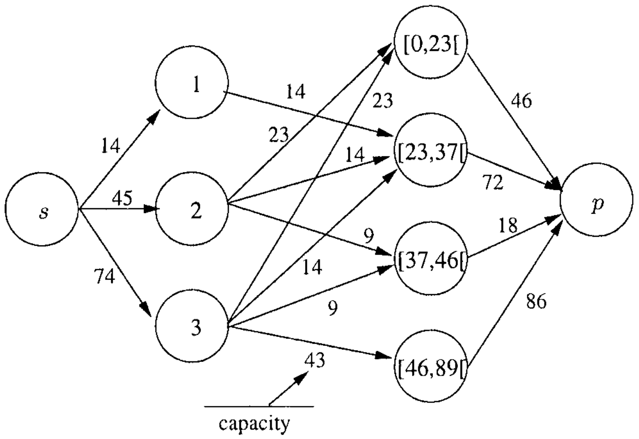


Figure 7.1. Graph for preemptive relaxation of π-machine problem

Finally, (Tercinet et al., 2004) prove that the semi-preemptive relaxation can be replaced by mixing energetic reasoning and preemptive relaxation. The approach is quite similar to the one proposed by Haouari and Gharbi but works of activities for time-interval (see 7.2.2.4, for formal description), that are the mandatory parts of the activities that must be processed during this time-interval, are used as minimum capacities on the edges between activities and corresponding time-intervals. Notice that the authors also prove that this approach may be better than the separate use of preemptive relaxation and energetic reasoning based satisfiability tests.

Jackson Pseudo Preemptive Schedule (JPPS). (Carlier and Pinson, 1998) present a lower bound for the identical parallel machine problem, called Jackson Pseudo Preemptive Schedule ($JPPS$), (see example of Figure 7.3, for the instance with 2 machines given in the table below), in which the preemption of any activity is also allowed, and in which we assume that a machine can be shared by several activities (see machine 1 on [4; 5]) and that an activity can be processed on several machines at a time (see activity 3 on [3; 4]).

Table 7.3. Instance of identical parallel machine problem

$i \in \mathcal{A}$	1	2	3	4	5
ES_i	0	1	2	3	3
p_i	3	2	3	1	1
q_i	4	4	1	0	0

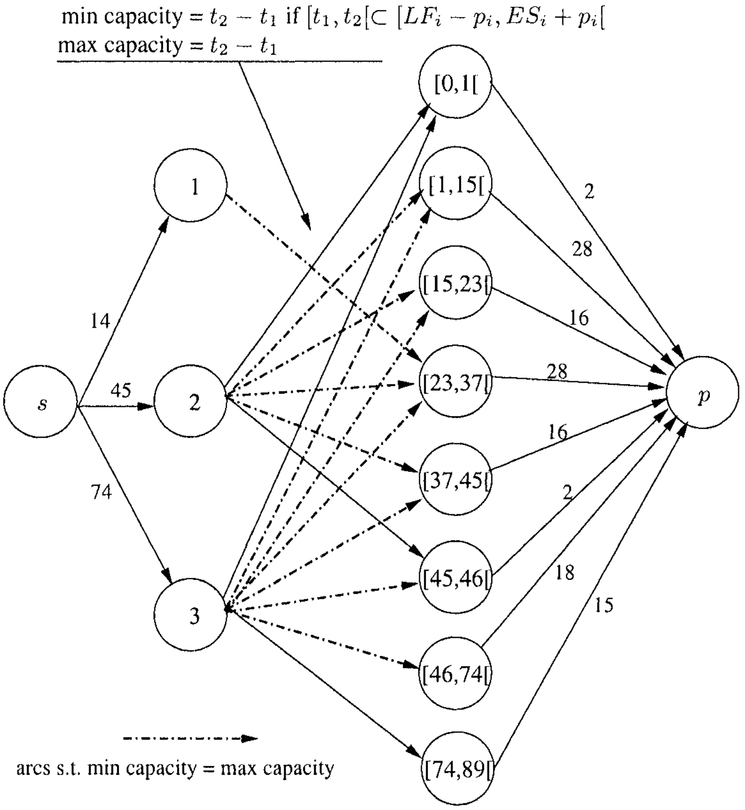

min capacity = $t_2 - t_1$ if $[t_1, t_2[\subset [LF_i - p_i, ES_i + p_i[$
max capacity = $t_2 - t_1$
s
1
2
3
14
45
74
[0,1[
[1,15[
[15,23[
[23,37[
[37,45[
[45,46[
[46,74[
[74,89[
p
2
28
16
28
16
2
18
15
arcs s.t. min capacity = max capacity

Figure 7.2. Graph for semi-preemptive relaxation of π-machine problem

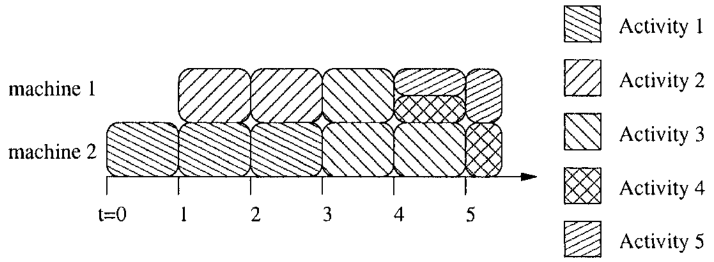


Figure 7.3. Example of JPPS for a 2-machine problem

Note that if deadlines rather than tails are associated with activities, we can rely on a destructive bound (see Introduction). Furthermore, (Carlier and Pinson, 2004) propose adjustments of release dates for this problem, that will

be not retranscribed in this chapter.
To build $JPPS$, a list scheduling algorithm is used whose priority dispatching rule at time t is the dynamic complete tail $c_i(t) = q_i + \xi_i(t)$, where $\xi_i(t)$ is the remaining processing time of activity i at time t in the current schedule. They divide the available activities in two classes: the *partially* available activities, whose part executed on $[ES_i; t]$ has been as large as possible ($\xi_i(t) = p_i - (t - ES_i)$), so that they can be processed at a rate $\alpha_i(t) \leq 1$, and the *totally* available activities, for which $\xi_i(t) > p_i - (t - ES_i)$, so that they can be processed at a rate $\alpha_i(t) \leq \pi$. Note that in this schedule we must respect at any time t, $\xi_i(t) \geq p_i - (t - ES_i)$.
The computation of the schedule blocks and of the next decision time are now presented. At time t, the algorithm schedules first the available activities with maximal complete tail at a maximal rate consistent with their status (partially or totally available). The events that can modify the current schedule block are one of the following types:

- a not in-process activity simply becomes available,
- an in-process activity is completed,
- a not in-process activity gets a higher priority than an in-process activity,
- a totally available activity, which is processed at a rate greater than 1, becomes partially available,
- a partially available activity, which is processed at a rate lower than 1, becomes totally available.

Several properties of JPPS are presented in (Carlier and Pinson, 1998) and (Carlier and Pinson, 2004). The makespan of JPPS can be computed in $O(n\ log\ n + n\ \pi\ log\ \pi)$ ($O(n^2 + n\ \pi^2)$, for building JPPS). So that it allows its intensive use in an enumerative process. JPPS does not systematically match the optimal preemptive solution. Then, the McNaughton schedule issued from JPPS where the activities have a rate lower than or equal to 1 at any time t, is not a lower bound of the problem.
(Carlier and Pinson, 1998) propose a simple adaptation of JPPS for the CuSP (see Section 7.2.2.3 for a formal description), without additional computational effort, that confirms the interest of JPPS for computing lower bounds for the RCPSP.

7.2.2.3 Deducing redundant Cumulative Scheduling Problem. This section is dedicated to techniques used to build redundant Cumulative Scheduling Problem (CuSP) from an initial RCPSP. These methods are based on the notion of Linear Lower Bound and Redundant Resources. Lower bound for CuSP is presented in the next section. The method that we present is based on a sub-problem where release dates and deadlines are not taken into account: we focus on specific linear forms that are lower bounds of the makespan when exactly part κ_i of the activity i has to be processed. These linear forms are used to compute Cumulative Scheduling Problem, that can be used to compute efficient lower bounds.

The CuSP is made up of a set $\mathcal{A}$ of activities that has to be processed on a single renewable cumulative resource of capacity R. Each activity, $i \in \mathcal{A}$, has a release date ES_i, a processing time p_i and a deadline LF_i, and it requires r_i units of the resource to be processed. These problems are known to be $\mathcal{NP}$-Hard in the strong sense. Getting relevant CuSP, considering that release dates and tails of activities are deduced from the precedence constraints of the initial RCPSP, is useful for bounding RCPSP: CuSP can be seen as an extension of identical parallel machine problem where activities need more than one machine to be processed.

Linear Lower Bound definition. A Linear Lower Bound (LLB) is based on the relaxation of the RCPSP instance to one resource Cumulative Scheduling Problem (CuSP). A CuSP can be obtained from a RCPSP and a trial value D by ignoring all the resources but one, and by relaxing the precedence constraints to release dates and deadlines of activities, computed according to precedence relationship.

DEFINITION 7.1 *The linear form:* $(p_1, \dots, p_n) \rightarrow \sum_{i \in \mathcal{A}} \lambda_i \cdot p_i$ *is a Linear Lower Bound if, for any* κ_i $(0 \leq \kappa_i \leq p_i)$, $\sum \lambda_i \cdot \kappa_i$ *is a lower bound of the makespan when exactly part* κ_i *of the activity* i *has to be processed. (Carlier and Néron, 2003).*

We present here some of the classical lower bounds that can be expressed as Linear Lower Bound.

The basic bound. $(1/R) \sum_{i \in \mathcal{A}} r_i \cdot p_i$ is a LLB of the makespan of the CuSP. This basic bound is very useful due to energetic reasoning (see 7.2.2.4) that can be applied with it for getting adjustments of release dates and deadlines. We will see in the next paragraph, that any LLB is the basic bound of the redundant resource it is associated with, which explains the interest of redundant resources.

The critical path. Let $\Gamma = \{i_1, i_2, \dots, i_r\}$ a path in the conjunctive graph of the initial RCPSP, that is: i_1 precedes i_2, i_2 precedes i_3 ..., etc. $\sum_{i \in \Gamma} p_i$, is a LLB. Moreover, if the path is optimal, it is equal to the value of the critical path.

The bound of Mingozzi et al. Let us recall the linear programming technique of Mingozzi et al (Mingozzi et al., 1998) in the case of a CuSP. Let J be a subset of activities that can be processed simultaneously, i.e., $\sum_{i \in J} r_i \leq R$. At first we associate with J the $\{0, 1\}$ column vector defined by $\psi_J(j) = 1$ if $j \in J$. The bound of Mingozzi et al is obtained by solving the linear program below where P is the vector of the processing times and τ the vector of the τ_J, (τ_J represents the duration in the schedule during which J is the subset of activities processed). Ψ is the matrix whose columns are the vector ψ_J

$$\min \sum_J \tau_J,\ \Psi \cdot \tau \geq P \wedge \tau \geq 0.$$

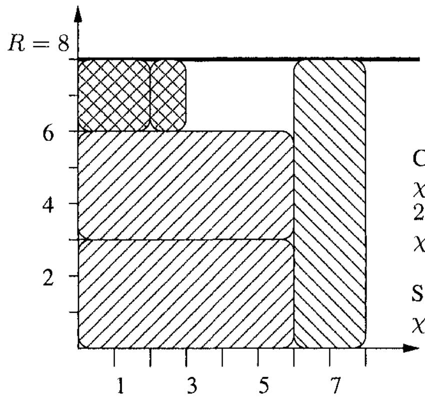
This bound can be improved by taking into account release dates and deadlines for defining J (Brucker and Knust, 2000). Indeed, this bound corresponds to the optimal makespan when preemption is allowed. The dual of the previous linear program is :

$$\max\ v^t \cdot P,\ v \cdot \Psi \leq 1 \wedge v \geq 0.$$

Let v satisfying $v \cdot \Psi \leq 1$, $v \geq 0$, then $v^T \cdot P$ is a LLB. Moreover when $v = v^*$ (the optimal solution of the dual), it is equal to the Mingozzi bound. The corresponding LLB is a very strong one, although it is very difficult to compute it due to the large size of the linear program. An interesting property is that the polyhedra of the dual is independent of P. This property is used to get the Multiple Elastic Preemptive Bound.

The multiple elastic preemptive bound (Carlier and Néron, 2000). In the Multiple Elastic Preemptive (MEP) Relaxation, preemption is allowed, and several parts of the same activity can be processed simultaneously. At first, all the activities having the same resource requirement are merged. Thus, P_r is defined as the sum of the processing times of the activities that require r units of the resource during their processing: $P_r = \sum_{i \in \mathcal{A}/r_i = r} p_i$. $Congif^R = \{c_1, c_2, ..., c_K; c_k \in \mathbb{N}^*\}$ is called a *feasible configuration* for R if: $\sum_{k=1}^{K} c_k \leq R$ and $c_1 \geq c_2 \geq ... \geq c_K$. $\{Config_h^R; h \in \{1, \ldots, H\}\}$ denotes the set of feasible configurations for a resource capacity of R units, and $Config_h^R[r]$ is the number of resource requirements equal to $r \in \{1, \ldots, R\}$ in $Config_h^R$. Note that the number H of feasible configurations depends on R and can be very large. Figure 7.4 presents a non optimal schedule illustrating these notions. To simplify the presentation we have only considered four configurations, which are reported on the figure. According to the configuration constraints, it can been stated (Carlier and Néron, 2000), that an optimal solution of the preemptive problem where several parts of the same activity can be processed simultaneously, is given by the solution of (7.1): χ_h is the duration of the configuration $Config_h^R$, and then $\sum_{h=1}^{H} \chi_h$ is the makespan.

$$\min \sum_{h=1}^{H} \chi_h \text{ s.t. } \forall r \in \{1, \ldots, R\}\ \sum_{h=1}^{H} Config_h^R[r] \cdot \chi_h \geq P_r\ ;\ \chi_h \geq 0 \quad (7.1)$$



$Config_1 = \{3, 2, 2\}$
$Config_2 = \{3, 3\}$
$Config_3 = \{8\}$

Constraints due to configurations
$\chi_1 = P_2$
$2 \cdot \chi_1 + 2 \cdot \chi_2 = P_3$
$\chi_3 = P_8$

Solution associated with the schedule
$\chi_1 = 3$, $\chi_2 = 3$, $\chi_3 = 2$.

Figure 7.4. A schedule for $R = 8$, $P_2 = 3$, $P_3 = 12$ and $P_8 = 2$

(7.2) is the corresponding dual formulation. Its matrix depends only on the value of R.

$$\max \sum_{r=1}^{R} P_r \cdot o_r \text{ s.t. } \forall h \in \{1, \ldots, H\} \sum_{r=1}^{R} Config_h^R[r] \cdot o_r \leq 1 \qquad (7.2)$$

(7.1) and (7.2) are solved in (Carlier and Néron, 2000) for any value of R smaller than or equal to 10, i.e, all corresponding $LLBs$ have been explicitly enumerated by hand. The method for solving this linear program is based on the enumeration of the optimal solutions of the dual linear program for any P.

Indeed, let $S_{opt}^g(R)$, $g \in \{1, \ldots, G\}$ denote the g-th optimal solution of the dual LP for a given value of R. Then its associated cost $MEPB^g(R)$ is a LLB on P_r. Let $\lambda_1^g \cdot P_1 + \lambda_2^g \cdot P_2 + \ldots + \lambda_R^g \cdot P_R$, $g \in \{1, \ldots, G\}$ be these LLB. Due to linear programming properties, $\max_g MEPB^g(R)$ is the optimal value of (7.1) and (7.2), and so a valid lower bound of the makespan. The main advantage of this MEP relaxation is to take into account, the idle time that may occur due to the configurations involved in addition of the global work. Moreover all the $LLBs$ are independent of P and can be tabulated, as explained in (Carlier and Néron, 2000).

Redundant functions and redundant resources. Recently the works dealing with the MEP relaxation, have been generalized to enumerate the sets of valid LLB. This enumeration is based on the notion of *redundant functions* and *maximal redundant functions* (MRFs), and some fundamental properties of these MRF. We now present the notion of Redundant Resources that is used to get relevant one-resource instances from one initial RCPSP instance restricted to one of its resources and one Redundant Function. Let us recall that r_i is the resource requirement of activity i.

DEFINITION 7.2 *A Redundant Function (RF) f is a discrete mapping of $\{0,\ldots,R\} \rightarrow \{0,\ldots,R'\}$ ($R \in \mathbb{N}, R' \in \mathbb{N}$) such that: $i_1 + i_2 + \ldots + i_k \leq R \Rightarrow f(i_1) + f(i_2) + \ldots + f(i_k) \leq R'$.*

RFs and LLBs are strongly linked: let $\sum_{i\in\mathcal{A}} a_i \cdot p_i$ be a LLB, and let us assume that all a_i are rational which $a_i = a'_i/R', \forall i \in \mathcal{A}$. Then an associated RF, f can be defined by $f : \{0,\ldots,R\} \rightarrow \{0,\ldots,R'\}, f(r_i) = a'_i$.

The authors present the link between this approach and the dual feasible solution introduced in (Johnson et al., 1974) for the bin-packing problem and used recently in (Fekete and Schepers, 1998), and (Martello and Toth, 1990).

DEFINITION 7.3 *Let f be a redundant function $\{0,\ldots,R\} \rightarrow \{0,\ldots,R'\}$. The associated redundant resource has a capacity of R'. Moreover, activity i needs $f(r_i)$ units of the redundant resource.*

So the basic bound on this redundant resource is now equal to $\sum_{i\in\mathcal{A}} \frac{f(r_i)}{R'} \cdot p_i$. This new bound can be larger than the one computed on the initial resource, which confirms the interest of Redundant Resources.

Table 7.4. An Example of Redundant Resources

$i \in \mathcal{A}$	1	2	3	4	5	6
r_i	3	5	1	4	2	2
p_i	2	1	1	1	1	2
$f_1 : \{0,\ldots,5\} \rightarrow \{0,\ldots,1\}, f_1(r_i)$	1	1	0	1	0	0
$f_2 : \{0,\ldots,5\} \rightarrow \{0,\ldots,2\}, f_2(r_i)$	1	2	0	2	1	1

Figure 7.5 shows how redundant resources are built for two MRFs.

Unfortunately, there are a lot of redundant functions for each couple (R, R'). To restrict the search to the most relevant ones we introduce the notion of *Maximal Redundant Function*: a Maximal Redundant Function (MRF) is a redundant function such that there exists no redundant function f' with $f' > f$, i.e., $\forall i \in \{0,\ldots,R\}, f'(i) \geq f(i) \Rightarrow f' = f$.

Both an enumeration scheme to compute all MRF for fixed (R, R') and a linear programming formulation to detect the ones that are dominated, have been proposed. The notion of non-dominated MRFs corresponds to MRFs that may lead to interesting lower bounds. The fact that f is a non-dominated MRF corresponds to the existence of a set of processing times of activities such that the best lower bound for these processing times is given by the redundant resource corresponding to f.

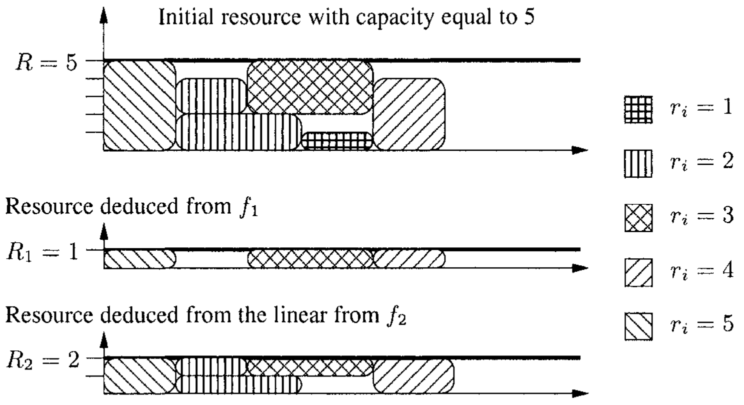

Initial resource with capacity equal to 5
$R = 5$
Resource deduced from f_1
$R_1 = 1$
Resource deduced from the linear from f_2
$R_2 = 2$
$r_i = 1$
$r_i = 2$
$r_i = 3$
$r_i = 4$
$r_i = 5$

Figure 7.5. Example of feasible schedules for redundant resources, R = 5

7.2.2.4 Energetic reasoning for Cumulative Scheduling Problem. At first, we present a brief overview of the energetic reasoning adapted to the CuSP. Energetic reasoning aims to develop satisfiability tests that are used to prove that there exists no solution in which activities are processed within their time-windows $[ES_i, LF_i]$, without violating the machine constraint. It can be used to compute destructive lower bound for the initial RCPSP instance. This approach has been originally developed by (Erschler et al., 1991) and (Lopez et al., 1992) to solve Cumulative Scheduling Problems. Recently (Baptiste et al., 1999), aim to develop satisfiability tests and time-bound adjustments to ensure that either a given schedule is not feasible or to derive some necessary conditions that every feasible schedule must satisfy. Further details and improvements can be found in (Schwindt, 2005).

Given a time-interval $[t_1, t_2]$, satisfiability tests are based on the computation of the part of the activity i, that must be processed between t_1 and t_2 where, without loss of generality, we assume that $t_1 < t_2$. The mandatory part of activity i is called its *work* in the time-interval $[t_1, t_2]$. To compute it, the activities are either *left-shifted* or *right-shifted* on their time-windows $[ES_i, LF_i]$, i.e., an activity either starts at ES_i or ends at LF_i (see Figure 7.6).

Using Figure 7.6, we formally define the work in interval $[t_1, t_2]$ as follows:

$$\begin{aligned}
W_{left}(i, t_1, t_2) &= r_i \cdot \min(t_2 - t_1, p_i, \max(0, ES_i + p_i - t_1)), \\
W_{right}(i, t_1, t_2) &= r_i \cdot \min(t_2 - t_1, p_i, \max(0, t_2 - LF_i + p_i)), \\
W(i, t_1, t_2) &= \min(W_{right}(t_1, t_2), W_{left}(t_1, t_2)) \\
&= r_i \cdot \min(t_2 - t_1, p_i, \max(0, ES_i + p_i - t_1), \\
&\qquad \max(0, t_2 - LF_i + p_i)).
\end{aligned}$$

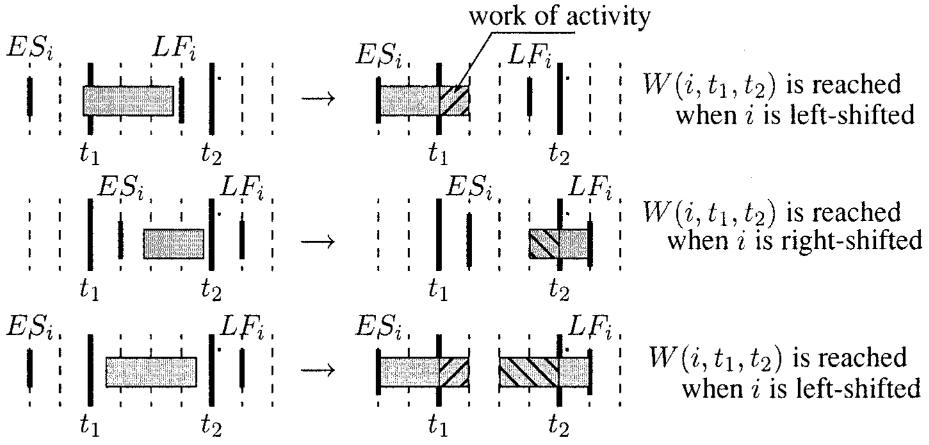

work of activity
ES_i
LF_i
t_1
t_2
$W(i, t_1, t_2)$ is reached when i is left-shifted
$W(i, t_1, t_2)$ is reached when i is right-shifted
$W(i, t_1, t_2)$ is reached when i is left-shifted

Figure 7.6. Work of an activity through $[t_1, t_2]$.

PROPERTY 7.4 **Satisfiability Test:** *If there exists some time interval* $[t_1, t_2]$, $\sum_{i \in \mathcal{A}} W(i, t_1, t_2) > R \cdot (t_2 - t_1)$ *then the CuSP has no solution.*

Based on the same idea, (Baptiste et al., 1999) propose a method to adjust the time-bounds of activities. We assume without loss of generality that the satisfiability tests have been done for all time-intervals. Let us now introduce $Sl(j, t_1, t_2)$, *the slack of* $j \in \mathcal{A}$ *over* $[t_1, t_2]$. Roughly speaking, this slack corresponds to the available *energy* that can be used to process j. Following the notation previously introduced, the slack can be defined as:

$$Sl(j, t_1, t_2) = \frac{R \cdot (t_2 - t_1) - \sum_{i \in \mathcal{A}, i \neq j} W(i, t_1, t_2)}{r_i}$$

If the right-work of the activity $j \in \mathcal{A}$ is strictly greater than $Sl(j, t_1, t_2)$, the activity cannot be right-shifted, i.e., it cannot end at its deadline. Hence, only a part of j, smaller than or equal to the slack $Sl(j, t_1, t_2)$ can be processed on $[t_1, t_2]$.

PROPERTY 7.5 **Release Date Adjustments.** *For any activity* $j \in \mathcal{A}$,

$$[W_{left}(j, t_1, t_2) > Sl(j, t_1, t_2)] \Rightarrow [ES_j \leftarrow \max(ES_j, t_2 - Sl(j, t_1, t_2))]$$

PROPERTY 7.6 **Deadline Adjustments.** *For any activity* $j \in \mathcal{A}$,

$$[W_{right}(j, t_1, t_2) > Sl(j, t_1, t_2)] \Rightarrow [LF_j \leftarrow \min(LF_j, t_1 + Sl(j, t_1, t_2))]$$

One crucial point to apply efficiently energetic reasoning is to determine what are the relevant time-intervals on which the above properties have to be applied. It is proved (see Baptiste et al., 1999) that for classical energetic reasoning there are only $O(n^2)$ relevant time-intervals, that are *the breaking points* of the work of activities depending on t_1 and t_2.

$$\begin{aligned}
t_1 &\in \{ES_i, i \in \mathcal{A}\} \cup \{LF_i - p_i, i \in \mathcal{A}\} \cup \{ES_i + p_i, i \in \mathcal{A}\} \\
t_2 &\in \Theta(t_1) \text{ with } \Theta(t) = ES_i + LF_i - t \\
t_2 &\in \{LF_i, i \in \mathcal{A}\} \cup \{ES_i + p_i, i \in \mathcal{A}\} \cup \{LF_i - p_i, i \in \mathcal{A}\} \\
t_1 &\in \Theta(t_2) \text{ with } \Theta(t) = ES_i + LF_i - t
\end{aligned}$$

7.3 Using constraint propagation to tighten the problem

This part is devoted to constraint programming based approaches that can be applied both to compute destructive lower bounds for the RCPSP and to adjust time-windows of activities. Three families of constraint propagation algorithms can be distinguished for dealing with renewable resources: *timetabling* techniques are based on the computation of an aggregation of the resource demand at every time-point; *edge finding* and *activity intervals* techniques rely on the analysis of the resource demand over time intervals whereas *conjunctive reasoning with temporal constraints* are based on an analysis of the current temporal constraint network. Those three families of techniques are described below more in detail.

7.3.1 Timetabling

Timetabling relies on the computation for every time-point t of the minimal resource usage at this time-point by the current activities in the schedule (Le Pape, 1994). This aggregated demand profile is maintained during the search and allows to restrict the domains of the start and end times of activities by removing the time-points that would necessarily lead to an over-consumption of the resource. Note that timetabling is also sometimes referred to as resource histograms (Caseau and Laburthe, 1996). Suppose a resource requirement $r_{i,k}$ on a given renewable resource k such that $LS_i < EF_i$, then we know surely that activity i will at least execute over the time interval $[LS_i, EF_i)$. Thus, it will surely require $r_{i,k}$ units of resource k all along this time interval. For each resource k, a curve $Req_k(t)$ is maintained that aggregates all these demands:

$$Req_k(t) = \sum_{i/LS_i \le t < EF_i} r_{i,k}$$

It is clear that if there exists a time-point t such that $Req_k(t) > R_k$, the current schedule cannot lead to a feasible solution. Thus if D is the trial value (see Introduction) that has been used to compute deadline then $D+1$ is a valid lower bound. Furthermore, if there exists a resource requirement $r_{i,k}$ and a time-point

t_0 such that:

$$EF_i \leq t_0 < LF_i \text{ and } \forall t \in [t_0, LF_i), Req_k(t) + r_{i,k} > R_k$$

then, activity i cannot end after time-point t_0. It would otherwise over-consume the resource. Indeed, remember that, as $EF_i \leq t_0$, i is never taken into account in the aggregation on the time interval $[t_0, LF_i)$. Thus, t_0 is a new valid upper bound for the end time of activity i. Similar reasoning can be applied to find new lower bounds on the start time of activities.

7.3.2 Edge finding and activities intervals

This section presents a variant of the edge-finding adjustments described in section 7.2.1.3 that works on cumulative machines rather than on one machine problem. Let $\Omega \subset \mathcal{A}$ a subset of activities that require a given resource k of capacity R_k. In addition to the notation introduced in section 7.2.1.3, let us denote:

- $EF_\Omega = \min_{j\in\Omega} EF_j$, the earliest finishing time of all activities in Ω, in a similar way, let us define $LF_\Omega = \max_{j\in\Omega} LF_j$, $ES_\Omega = \min_{j\in\Omega} ES_j$, and $LS_\Omega = \max_{j\in\Omega} LS_j$
- $w_\Omega = \sum_{j\in\Omega} p_j \cdot r_{j,k}$, the global energy required by Ω (on resource R_k).

The basic idea of edge-finding and activity interval techniques is to ensure that for any subset of activities Ω, resource R_k provides enough energy over the time interval $[ES_\Omega, LF_\Omega)$ to allow the execution of all the activities of Ω, that is: $w_\Omega \leq R_k \cdot (LF_\Omega - ES_\Omega)$. Constraint propagation is usually performed by applying the three following deduction rules, where $i \in \mathcal{A} \setminus \Omega$:

- If $R_k \cdot [LF_\Omega - ES_{\Omega\cup\{i\}}] < w_{\Omega\cup\{i\}}$, then i must finish after all activities in Ω, in particular $EF_i \leftarrow \max(EF_i, EF_\Omega)$.
- If $ES_\Omega < ES_i < EF_\Omega$ and $R_k \cdot [LF_\Omega - ES_\Omega] < w_\Omega + r_{i,k} \cdot [\min(LF_\Omega, EF_i) - ES_\Omega]$, then at least one activity j in Ω must precede i, in particular $ES_i \leftarrow EF_\Omega$.
- If $ES_i < ES_\Omega < EF_i$ and $R_k \cdot [LF_\Omega - ES_\Omega] < w_\Omega + r_{i,k} \cdot [EF_i - ES_\Omega]$, then i must finish after all activities in Ω, in particular $EF_i \leftarrow \max(EF_i, EF_\Omega)$.

These rules allow to update the earliest start or completion times of activities and symmetrical rules allow to update the latest start and completion times. Edge finding (Nuijten, 1994) and activity intervals propagation (Caseau and Laburthe, 1996) are very similar techniques that mainly differ in the way the propagation rules are triggered: edge-finding algorithms are global algorithms that perform all updates on a given resource whereas activity interval approaches are performed incrementally as soon as the time bound of an activity changes.

The propagation of edge-finding and activity interval can be strengthened by considering other time intervals than the one related with the time-bounds of activities as described in 7.2.2.4.

7.3.3 Distances between activities and shaving

(Brucker and Knust, 2000) and recently (Demassey et al., 2005) take advantage of the deductions performed by constraint propagation to preprocess linear programs, by fixing variables and strengthening constraints. In this paragraph B is the distance matrix between all couple of activities, and $b_{i,j}$ denotes the minimal distance between i and j (*i.e.* $S_j - S_i \geq b_{i,j}$)

The CP algorithm includes the classical filtering techniques described such as edge-finding, immediate selection (both are described in the previous subsection) and symmetric triples rules, but is run on an alternative CSP formulation based on sequencing variables: for each couple of activities (i, j), variable $X_{i,j}$ denotes the difference between the starting times S_i and S_j. Such a variable corresponds to the *distance* between activities i and j in the activity-on-node graph. Its domain is an interval $[b_{i,j}, -b_{j,i}]$. By ensuring *bound consistency*, constraint propagation allows to increase values $b_{i,j}$ but also to identify new *disjunctions* $i - j$ which are couples of mutually incompatible activities (i.e. forbidden sets of size 2, $\mathcal{F}_2$).

Additionally, Demassey et al perform global filtering with a suited *shaving* technique, which follows the general principle of consistency enforcing techniques based upon probing: a new constraint c is temporarily added and constraint propagation is performed. If it leads to an infeasibility, then the opposite constraint $\rceil c$ is consistent with the problem. In this implementation, the validity of the three following constraints is tested, for each pair of activities $\{i,j\}$: $i \longrightarrow j$ (j follows i), $j \longrightarrow i$ (i follows j) and $i \parallel j$ (i and j are both executed in parallel during at least one time period). Shaving aims to prove if such sequencings are infeasible or necessary. To achieve this goal, it uses intermediate results that may be very helpful to deduce informations on the problem. For example, after fixing constraint $i \longrightarrow j$, constraint propagation computes the minimal distance matrix $B^{i \rightarrow j}$ and a set $\mathcal{F}_2^{i \rightarrow j}$ of disjunctions that can be identified among all the feasible schedules such that i precedes j. Some of these informations can easily be exploited to derive valid linear inequalities.

The approach of (Demassey et al., 2005) which is described in section 7.4.2.1, is successfully applied to two well-known integer linear formulations for the RCPSP.

7.3.4 Conjunctive reasoning with temporal constraints

The propagation algorithms described above only reason on the time bounds of activities (ES_i, LS_i, EF_i, LF_i) and do not directly take into account the precedence constraints that may exist between activities. Some constraint propagation algorithms have been recently proposed that exploit the current temporal constraint network and proved to be very efficient especially when used in conjunction with a branching scheme that solves the scheduling problem by adding precedence constraints (see section 7.4.1).

These algorithms require that a temporal network representing the relations between the time-points (start and end) of all activities using the point algebra of (Vilain and Kautz, 1986) is maintained during the search. We denote $\{\emptyset, \prec, \preceq, =, \succ, \succeq, \neq, ?\}$ the set of qualitative relations between time points. The temporal network is in charge of maintaining the transitive closure of those relations. If S_i and C_i respectively denote the start and end time-point of activity i, the initial set of relations consist of the precedences $S_i \prec C_i$ for each activity i and $C_i \preceq S_j$ for each precedence constraint $(i, j) \in E$. During the search additional precedence relation can be added as decisions or as the result of constraint propagation.

The *energy precedence* propagation (Laborie, 2003) for an activity i on a resource k ensures that for each subset Ω of predecessor activities of activity i the resource provides enough energy to execute all activities in Ω between ES_Ω and S_i. More formally, it performs the following deduction rule:

$$\forall \Omega \subset \{j \in \mathcal{A}, C_j \preceq S_i\}, ES_i \leftarrow max(ES_i, ES_\Omega + \lceil w_\Omega / R_k \rceil)$$

The propagation of the energy precedence constraint can be performed for all the activities i, on a resource and for all the subsets Ω with a total worst-case time complexity of $O(n(p + log\ n))$ where n is the number of activities on the resource and p the maximal number of predecessors of a given activity in the temporal network ($p < n$).

On a renewable resource, the *balance constraint* (Laborie, 2003) can be defined as follows. The basic idea of the algorithm is to compute, for each activity i on a resource k, a lower bound on the resource usage at the start time of i (a symmetrical reasoning can be applied to perform some propagation based on a lower bound on the resource usage at the completion time of i). Using the temporal network a lower bound on the resource utilization at time-point $S_i + \epsilon$, i.e., just after the start time of i can be computed assuming that all the activities requiring the resource that do not necessarily overlap S_i will not overlap it:

$$L_k(i) = \sum_{j/(S_j \preceq S_i) \wedge (C_j \succ S_i)} r_{i,k}$$

Given this bound, the balance constraint is able to discover three types of information:

- **Dead ends.** Whenever $L_k(i) > R_k$, the resource will surely be over-consumed just after time-point S_i so the search has reached a dead end.

- **New bounds on time variables.** If $L_k(i) \leq R_k$, $\Delta(i) = R_k - L_K(i)$ represents a slack of capacity that must not be exceeded by all the resource requirements that, currently, do not necessarily overlap S_i but could overlap it. Let $\Pi(i) = \{j/(S_j \preceq S_i) \wedge \neg(C_j \succ S_i)\}$. We suppose the activities $(j_1, \ldots, j_u, \ldots, j_p)$ in $\Pi(i)$ are ordered by decreasing earliest completion time EF_j. Let v be the index in $\{1, \ldots, p\}$ such that:

$$\sum_{u=1}^{v-1} r_{j_u,k} \leq \Delta(x) < \sum_{u=1}^{v} r_{j_u,k}$$

  If event S_i occurs at a time-point $S_i < EF_{j_v}$, not enough activity will be able to be completed strictly before S_i in order to ensure the resource is not over-consumed just after S_i as in this case, the consumed quantity will be at least $L_k(i) + \sum_{u=1}^{v} r_{j_u,k} > R_k$. Thus, EF_{j_v} is a valid lower bound of S_i.

- **New precedence relations.** Suppose that there exists activity h in $\Pi(i)$ such that:

$$\sum_{l \in \Pi(i), C_l \succeq C_h} r_{l,k} > \Delta(x)$$

  Then, if we had $S_i \prec C_h$, we would see that again there is no way to avoid a resource over-consumption as it would consume at least:

$$L_k(i) + \sum_{l \in \Pi(i), C_l \succeq C_h} r_{l,k} > R_k$$

  Thus, the necessary precedence relation: $C_h \preceq S_i$ can be deduced and added to the current temporal network.

The balance algorithm can be executed for all the activities i with a global worst-case complexity in $O(n^2)$ if the propagation that discovers new precedence relations is not turned on, in $O(n^3)$ for a full propagation. In practice, there are many ways to shortcut this worst case and, in particular, it is noticed that the algorithmic cost of the extra-propagation that discovers new precedence relations is in general negligible.

7.4 Multi-resource based lower bound

In this Section, we describe lower bounds based on relaxations that take the multi-resource context into account. They are based on several possible representations of the resource constraints. In Section 7.4.1, the forbidden set representation is discussed and lower-bounds based on this structure are introduced. In Section 7.4.2, the resource constraints are tackled explicitly by different integer linear programming formulations. The lower-bounds are then derived by LP relaxation of these programs and cooperation between constraint programming and cutting-plane generation. Last, in Section 7.4.3, linear and constraint programming are used to compute lower bounds on the basis of the feasible configuration representation of the resource constraints.

7.4.1 Using forbidden sets for bounding RCPSP

7.4.1.1 Forbidden sets. A subset of activities can be considered as a forbidden set (also called minimal critical set) if the activities could be executed simultaneously and if there exists a resource k such that the sum of resource requirement of these activities for resource k over-consumes the resource (Bartusch et al., 1988). Forbidden sets are a simple generalization to cumulative scheduling of the pairs of activities competing for the same unary resource in disjunctive scheduling. If ϕ is a subset of activities, we denote $r_k(\phi) = \sum_{i\in\phi} r_{i,k}$ the global consumption of resource k by ϕ.

DEFINITION 7.7 (FORBIDDEN SET) *A forbidden set on a resource k is a subset of activities $\phi \subseteq \mathcal{A}$ such that:*
1. $R_k < r_k(\phi)$
2. $\forall\varphi \subsetneq \phi, r_k(\varphi) \leq R_k$
3. $\forall_{(i,j)\in\phi\times\phi} S_i \prec C_j$ is consistent with the current temporal network

Informally, the different ways to resolve a forbidden set consist in fixing a precedence constraint between any two of its activities.

DEFINITION 7.8 (RESOLVERS OF A FORBIDDEN SET) *If $\phi \subseteq \mathcal{A}$ is a forbidden set, the resolvers of ϕ consist of the set of temporal constraints $Res(\phi) = \{C_i \preceq S_j : (i,j) \in \phi\times\phi, i \neq j\}$.*

Forbidden sets can be exploited to compute lower bounds on the RCPSP considering a relaxation of the problem or directly be used in a complete search.

7.4.1.2 Relaxations based on forbidden sets. Starting from the remark that most of the forbidden sets in the hard instances of the PSPLIB benchmark (Kolisch (1996)) are of size 2 or 3, some relaxations are proposed in (Garaix et al., 2005) that only take into account those forbidden sets.

More precisely, when only forbidden sets of size 2 are considered (disjunctive relaxation $P(\mathcal{F}_2)$) and all the other ones are relaxed, the problem can be reformulated using unary resources only (disjunctive resources of capacity 1). Each maximal clique in the graph whose edges represent the forbidden sets corresponds to a unary resource in the relaxation. This relaxed problem can be solved using a complete classical disjunctive search. If the relaxed problem is shown to be unfeasible for a maximal makespan D, then, it means that $D+1$ is a legal lower bound of the original RCPSP.

This relaxation can be made tighter by considering forbidden sets of size 3 (relaxation $P(\mathcal{F}_3)$) and using a resource of capacity 2. Each activity in a forbidden set of size 3 requires 1 unit of the resource. From a modeling perspective, this resource of capacity 2 is represented as two unary resources and each activity in the forbidden set requires one of the two possible unary resources. In practice, as the number of forbidden sets of size 3 is very large, these forbidden sets are used as successive cuts to tighten the relaxation: a first try is performed using the disjunctive relaxation $P(\mathcal{F}_2)$ and in case a solution is found, only the forbidden sets of size 3 that are violated in this solution are added to the relaxed problem formulation. This process is repeated until a time limit is reached.

This approach allows improving several lower-bounds of the KSD instances reported on the PSPLIB web page together with some improvement reported in (Baptiste and Demassey, 2004) on the instances with 60 activities. The approach described in (Garaix et al., 2005) improves 13 lower bounds for the instances with 60 activities and 26 lower bounds for the instances with 90 activities.

7.4.1.3 Complete search based on forbidden sets. In the approach described in (Laborie, 2005), a complete search tree exploration is performed by selecting at each search node a forbidden set ϕ and branching on its possible resolvers in the children nodes until there is no more forbidden sets. This approach is clearly complete and can be used to compute lower bounds on the RCPSP by performing a complete search to prove the infeasibility of a particular makespan value and then used to compute destructive lower bounds (see Introduction).

As described in (Laborie and Ghallab, 1995), the set of resolvers $Res(\phi)$ of a forbidden set ϕ can be simplified so as to remove those resolvers $\rho \in Res(\phi)$ for which there exists another resolver $\rho' \in Res(\phi)$ such that $\rho \Rightarrow \rho'$ given the current temporal network. Indeed, in such case, the resolver ρ is redundant. In what follows, it is assumed that the set of resolvers of a forbidden set has been simplified.

As all the resolvers consist of temporal constraints of the form $\mathcal{T}_1 \preceq \mathcal{T}_2$ where $\mathcal{T}_1$ and $\mathcal{T}_2$ are two time-points (variable start or completion time of an activity), the estimation of the size of the search space after posting such a precedence constraint is particularly interesting to choose which forbidden set to solve at a given search node. The fraction of the search space that is preserved when adding a precedence constraint is estimated using the complementary of the commitment measure introduced in Laborie, 2003.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two time-points with respective lower and upper bound for time value: $[\mathcal{T}_1^{min}, \mathcal{T}_1^{max}]$ and $[\mathcal{T}_2^{min}, \mathcal{T}_2^{max}]$. The size of the search space is estimated by the Cartesian product of the domain of the two variables, that is, the area of the rectangle $[\mathcal{T}_1^{min}, \mathcal{T}_1^{max}], [\mathcal{T}_2^{min}, \mathcal{T}_2^{max}]$. The size of the search space that is preserved when adding the constraint $\mathcal{T}_1 \preceq \mathcal{T}_2$ is the part of that rectangle above the line $\mathcal{T}_1 = \mathcal{T}_2$. The fraction of the search space that is preserved ($preserved(\mathcal{T}_1 \preceq \mathcal{T}_2)$) can thus be estimated as the ratio between those two areas.

If ω is the size of the search space below the current search node, the size of the search space after posting a temporal constraint $\mathcal{T}_1 \preceq \mathcal{T}_2$ can be estimated by $\omega \cdot preserved(\mathcal{T}_1 \preceq \mathcal{T}_2)$. If ϕ is the forbidden set that is selected to be resolved at the current search node, the size of the search space to explore below the current node can thus be estimated as the sum of the sizes of the search space below each child node, that is: $\omega \cdot \sum_{\rho \in Res(\phi)} preserved(\rho)$. Therefore, $preserved(\phi) = \sum_{\rho \in Res(\phi)} preserved(\rho)$ estimates the fraction of the search space that is preserved when choosing ϕ as the next forbidden set to solve. The heuristic proposed in (Laborie, 2005) chooses to resolve next the forbidden set ϕ^* that minimizes $preserved(\phi)$ that is, the one that minimizes the estimation of the size of the explored search space. Once such a forbidden set has been selected, it is simplified and the search explores all of its resolvers $\rho \in Res(\phi^*)$ in the child nodes by decreasing order of $preserved(\rho)$. This order has no effect when the schedule is not feasible as in this case the complete search tree needs to be explored but it helps finding a solution quicker when a solution exists.

The above approach is implemented on top of ILOG SCHEDULER 6.1 using the *timetable*, *edge-finding*, *precedence energy* and *balance* constraints described in section 7.3. The approach is benchmarked on the KSD instances with 60, 90 and 120 activities (Kolisch (1996)) using the lower and upper bound reported on the PSPLIB web page as of May, 1st, 2005 together with some improvement reported in (Baptiste and Demassey, 2004) on the instances with 60 activities. Within a time-limit of 1800s CPU time, out of the 617 previously open instances, 197 lower-bounds are improved (that is more than 31% of the previously open instances) and 97 instances are closed (that is more than 15% of the previously open instances). Furthermore, for all the 943 previously closed instances but two, the approach finds and proves the optimal solution. On the instance set with 60 activities, the critical path lower bound is improved by 8.57% in average, on the instance set with 90 activities this lower bound is improved by 7.40% in average and on the instance set with 120 activities by 22.97%.

The same approach is used to close with a time-limit of 5s CPU time all the open instances of the open-shop benchmark of (Guéret and Prins, 1999).

7.4.2 Linear programming relaxations and cutting plane generation

Earliest exact solution methods for the RCPSP were mainly branch-and-bound procedures based on linear programming formulations of the problem. A lower bound is usually obtained by omitting the resource constraints and then by computing the longest path problem in the precedence graph. In order to improve this lower bound, some research has been carried out to solve less drastic linear program relaxations.

Unfortunately, the resource constraints for the RCPSP are not only hard to handle but also hard to model as linear inequalities. For example, the well-known time-indexed linear formulations for the RCPSP mostly contains numerous variables and have poor linear relaxations. In order to enhance such a linear relaxation, some cutting-planes were previously described by (Christofides et al., 1987) and (Sankaran et al., 1999). Recent approaches integrate linear relaxations, cutting plane generation and constraint programming to derive strong lower bounds.

7.4.2.1 Time-Indexed Linear Formulation. The most encountered integer linear formulation of the RCPSP was first given by Pritsker et al (Pritsker et al., 1969) and is based on time-indexed 0-1 variables of the type: $y_{j,t} = 1$ if activity j starts at time t and $y_{j,t} = 0$ otherwise.

$$\min \quad \sum_{t=0}^{T} t \cdot y_{(n+1),t} \qquad \text{(D0)}$$

$$s.t. \quad \sum_{t=0}^{T} y_{j,t} = 1 \qquad \forall\, j \in \mathcal{A} \qquad \text{(D1)}$$

$$\sum_{t=0}^{T} t \cdot (y_{j,t} - y_{i,t}) \geq p_i \qquad \forall\, (i,j) \in E \qquad \text{(D2)}$$

$$\sum_{j \in \mathcal{A}} r_{j,k} \sum_{t_0 = t - p_j + 1}^{t} y_{j,t_0} \leq R_k \qquad \forall\, k \in \mathcal{R}, \forall\, t \in \{0, \ldots, T\} \qquad \text{(D3)}$$

$$y_{j,t} \in \{0,1\} \qquad \forall\, j \in \mathcal{A}, \forall\, t \in \{0, \ldots, T\} \qquad \text{(D4)}$$

Constraints (D1) state that each activity must be started exactly once over the planning horizon T. Inequalities (D2) and (D3) represent precedence and resource constraints, respectively. The size of this formulation is proportional to the value of the time horizon T and then may be very large. Furthermore, the optimal solutions of its linear relaxation are usually very fractional and their values give then weak lower bounds.

Preprocessing. To speed up the resolution of such a program, a good preprocessing is necessary. An efficient constraint propagation algorithm allows for example to drastically reduce its size by fixing numerous variables. Indeed, for each activity j in $\mathcal{A}$, a variable $y_{j,t}$ has to be defined only if t is a possible starting time for j. Hence, variables $y_{j,t}$ can be fixed to 0 for any t lower than the earliest starting time $ES_j = b_{0,j}$ of j, or greater than its latest starting time $LS_j = -b_{j,0}$.

Since $S_j = \sum_{t=ES_j}^{LS_j} t \cdot y_{j,t}$, precedence constraints may also be enhanced by taking into account the known minimal distances:

$$\sum_{t=ES_j}^{LS_j} t \cdot y_{j,t} - \sum_{t=ES_i}^{LS_i} t \cdot y_{i,t} \geq b_{i,j} \quad \forall\, (i,j) \in \mathcal{A}^2$$

Disaggregated precedence constraints. In (Christofides et al., 1987), Christofides et al introduce a *disaggregated* variant of the precedence constraints. Constraints (D2) can be replaced by:

$$\sum_{t_0=t}^{LS_i} y_{i,t_0} + \sum_{t_0=ES_j}^{t+b_{i,j}-1} y_{j,t_0} \leq 1 \, \forall\, (i,j) \in \mathcal{A}^2 \, \forall t \in \{ES_j - b_{i,j} + 1, \ldots, LS_i\} \tag{D2$_d$}$$

Despite of their larger number, these inequalities have two advantages. On one hand, they are together tighter than constraints (D2), giving then enhanced linear relaxation. On the other hand, the linear program obtained by replacing constraints (D2) by (D2$_d$) and by dropping resource constraints (D3) has the unimodularity property: its optimal fractional solutions are then integer.

Clique cuts. Clique cuts are well-known packing inequalities stating that if C is a maximal set of mutually incompatible activities (clique of disjunctions) then, at any time t, at most one activity of C is in process. Since additional disjunctions and conjunctions are likely to be detected by constraint programming, these clique cuts are expected to be stronger, after such a preprocessing, than the one used in classical implementations.

Shaving cuts. In this LP model, many informations deduced by shaving may be translated as linear inequalities. For example, the following relation is obviously valid: $S_j - S_i \geq p_i \implies S_l - S_h \geq b_{h,l}^{i \to j}$ ($b_{h,l}^{i \to j}$ is the distance between h and l if $i \to j$ is fixed). Furthermore, it is not dominated if the relative sequencing between activities i and j is yet unknown and if sequencing j after i improves the minimal distance between h and l. As for the precedence constraints (D2) and ($D2_S$), such a relation can be written according to both formalisms, aggregated or disaggregated. In the aggregated way, this can be modeled by the following inequality:

$$(-b_{j,i}-p_i+1)(\sum_{t=0}^{T} t\cdot(y_{l,t}-y_{h,t})-b_{h,l}) \geq (\sum_{t=0}^{T} t\cdot(y_{j,t}-y_{i,t})-p_i+1)(b_{h,l}^{i\to j}-b_{h,l}).$$

7.4.2.2 Computational experiments. We illustrate the quality of the lower bounds that can be obtained with the time indexed linear programming formulation by reporting the results of the constraint propagation based cutting plane procedure of (Demassey et al., 2002, Demassey et al., 2005) and the lagrangean relaxation approach proposed by (Möhring et al., 2003) on the 480 instances of (Kolisch (1996)) with 60 activities which is the smaller instance set with still open instances.

As reported by (Möhring et al., 2003) with their experiments on a Sun Ultra 2 with 200 MHz clock pulse and 512 MB of memory, solving the aggregated discrete LP relaxation without CP preprocessing nor cuts takes in average 3 seconds (279 seconds max) and improves on average the critical path lower bound by 5.2%. Adding the clique cuts gives an average deviation of 5.54% with an average CPU time of 6.4 seconds. To speed up the resolution of the linear relaxation (Möhring et al., 2003), following (Christofides et al., 1987), propose to dualize the resource constraints so as to obtain a lagrangian subproblem equivalent to a project scheduling problem with start-time dependent cost. They show that such a problem can be solved in polynomial time by minimum cut computations. Using a standard subgradient optimization to compute the optimal lagrangian multpliers, they reduce the computational time to 1.7 second in average (35 seconds max) without any loss of quality. (Demassey et al., 2002) show the benefit of incorporating constraint programming preprocessing and the above described shaving cuts by obtaining a deviation of 6.73% with, as a counterpart, an important increase of CPU times (45 seconds on average on an Pentium III cloked at 800 MHz). The results are still improved in Demassey et al., 2005 reaching a deviation of 7.72% by computing the lower bound in a destructive way. The CPU times however increase to 168 seconds on average (1963 max).

7.4.2.3 Continuous-time linear formulation. The classical Balas disjunctive formulation for the job-shop problem (Balas, 1970) is based on variables S_j modeling the starting time of activities j and mainly on disjunction variables $x_{i,j}$, stating if activity j starts or not after the completion of activity i. This formulation was extended to the RCPSP by Alvarez-Valdés and Tamarit (Alvarez-Valdés and Tamarit, 1993) making use of the concept of *forbidden sets*

$\mathcal{F}$ (cf. definition 7.7):

$$
\begin{array}{lllr}
\min & S_{n+1} & & \text{(C0)} \\
s.t. & x_{i,j} = 1 & \forall\,(i,j) \in E & \text{(C1)} \\
 & x_{i,j} + x_{j,i} \leq 1 & \forall\,(i,j) \in \mathcal{A}^2 & \text{(C2)} \\
 & x_{i,k} \geq x_{i,j} + x_{j,k} - 1 & \forall\,(i,j,k) \in \mathcal{A}^3 & \text{(C3)} \\
 & S_j - S_i \geq -M + (p_i + M)x_{i,j} & \forall\,(i,j) \in \mathcal{A}^2 & \text{(C4)} \\
 & \sum\limits_{i,j \in F} x_{i,j} \geq 1 & \forall\, F \in \mathcal{F} & \text{(C5)} \\
 & x_{i,j} \in \{0,1\}, S_i \geq 0 & \forall\,(i,j) \in \mathcal{A}^2 & \text{(C6)}
\end{array}
$$

Constraints (C1) give the precedence relations within the project. Constraints (C2) and (C3) ensure that no cycle will occur. Constraints (C4) model implications $x_{i,j} = 1 \Rightarrow S_j \geq S_i + p_i$ where M is a constant large enough to let S_i and S_j unrelated when $x_{i,j} = 0$. The resource constraints (C5) state that in any minimal forbidden set F, at least one sequencing decision must be taken.

Sequencing constraints and cuts. The implementation of this program is not realistic in general because of the possible exponential number of constraints (C5). In (Demassey et al., 2005), the authors tackle a relaxation of this program by dropping integrality constraints (C6) as well as constraints (C5) corresponding to minimal forbidden sets of cardinal strictly greater than 3. Nevertheless, their preprocessing algorithm described above, allow to identify much forbidden sets of size 2 which can directly be linearized as constraints (C5).

The CP formulation used in (Demassey et al., 2005) is close to this LP formulation. Indeed, distances $b_{i,j}$ are directly related to variables $x_{i,j}$:

$$
\begin{array}{lcll}
b_{i,j} \geq p_i & \Rightarrow & x_{i,j} = 1 & (j \text{ starts after the completion of } i) \\
b_{j,i} \geq 1 - p_i & \Rightarrow & x_{i,j} = 0 & (j \text{ starts before the completion of } i)
\end{array}
$$

Such conditions are then useful to fix variables in the linear program.

Some shaving informations can also be efficiently used to derive valid linear inequalities. For instance, condition $b_{h,l}^{i \to j} \geq p_h$ means that l follows h in any feasible schedules such that j follows i. In turn, such a relation may be formulated by the valid linear inequality:

$$x_{h,l} \geq x_{i,j} \qquad \forall (i,j,h,l) \in \mathcal{A}^4 \mid b_{h,l}^{i \to j} \geq p_h.$$

Distance constraints and cuts. Another lack of the continuous-time formulation is the presence of "big-M" values which are well known to provide poor relaxations. Here again, the precomputed minimal distances are good estimations to refine M values, since M can obviously be replaced by $-b_{i,j}$ in inequality (C4).

According that the optimal schedule duration is the length of a path made of arcs (i,j) such that $x_{i,j} = 1$, it is tempting to generate "path cuts" of type:

$$S_l - S_i \geq \alpha + \beta x_{i,j} + \gamma x_{j,l} \quad \forall (i,j,l) \in \mathcal{A}^3$$

where coefficients α, β, γ correspond to default evaluations of the distance $S_l - S_i$ according to the different possible values of x_{ij} and x_{jl}. Such evaluations can easily be deduced from distances b_{il} computed by shaving on the sequencing of (i,j) and (j,l). Demassey et al described how to obtain the deepest 3- or 4-activity path cuts according to any given evaluations.

Edge-finding cuts. Another kind of linear inequalities is closely related to the edge-finding rules described in section 7.2.1.3. They are also defined for any cliques C of activities pairwise in disjunction and aim at updating the starting time of one activity $j \in C$ with respect to the other activities in the clique. Inequalities of that kind have already been proposed for the job-shop problem by Dyer and Wolsey (Dyer and Wolsey, 1990) and Applegate and Cook (Applegate and Cook, 1991) and can easily be adapted to the RCPSP. Demassey et al proposed two variants based on the minimal distances in place of the earliest/latest starting times.

$$S_j \geq S_l + \sum_{i \in C \setminus \{j\}} p_i \cdot x_{i,j} + \sum_{i \in C \setminus \{l\}} b_{l,i} \cdot x_{i,l} \quad \forall j,l \in C$$

As an example, the above cut states that the distance between two activities j and l of clique C is greater than the sum of the durations of the activities in C which are scheduled between j and l.

7.4.2.4 Computational experiments. The linear programming relaxation of the continuous time formulation is known to be extremely poor if the cuts are not used. The only experiments using this formulation are reported by (Demassey et al., 2005) on 264 non trivial KSD instances with 30 activities, i.e., on whose the critical path lower bound is not feasible. The lower bound is computed in a constructive way. The complete constraint propagation process, including shaving, yields a lower bound of 3.6% from the optimal solution on average and finds 155 optimal solutions. The cutting plane generation procedure on the continuous time formulation reduces the gap to 3.2% and finds 160 optimal solutions. In comparison, the discrete time formulation without cuts reduces the gap exactly to the same value.

7.4.3 Feasible configuration and linear programming methods

A subset of $\mathcal{A}$ is said to be *feasible* if neither precedence nor resource constraints are violated.

DEFINITION 7.9 *A subset a of $\mathcal{A}$ is a* feasible subset *if and only if:*

1 *Resource constraints:* $\sum_{i \in a} r_{i,k} \leq R_k$, $\forall k \in \mathcal{R}$.

2 *Precedence constraints: for all pairs* $(i, j) \in a \times a$, *there is no path in* G *from* i *to* j *or from* j *to* i.

Activities of a feasible subset can be processed simultaneously. As described in Introduction, if we pretend to have an upper bound D of the makespan, constraint propagation techniques can be used to detect the infeasibility of any schedule of makespan lower or equal to D, and also to reduce the set A of all feasible subsets, as these techniques create new disjunctions between activities.

7.4.3.1 Initial formulation and relaxation. In (Mingozzi et al., 1998) a new exact formulation of the RCPSP is presented. This formulation uses $0-1$ discrete variables $\zeta_{i,t}$ that equal 1 if and only if activity i starts at time t, and $y_{l,t}$ that equal 1 if and only if the activities of the feasible subset a_l are in execution on time period $[t; t+1[$. Thus, precedence and no-preemption constraints can be simply formulated in spite of a very large number of variables (depending on the value D and on the cardinality of A). The resource constraints are included in the calculation of the feasible subsets. This formulation is used to derive new lower bounds.

By evading the non-preemption constraints and partially the precedence constraints (treated as disjunctions), (Mingozzi et al., 1998) also formulate several interesting LP-relaxations of the RCPSP. To do that, they introduce the continuous variable z_l which represents the total amount of time of execution of the activities of a_l in parallel, $z_l = \sum_{t=0}^{D} y_{l,t}$. One of their relaxation called (M) is presented below. Let us first define the subset A_{MAX} of the "undominated feasible subsets" of A such that $a_l \in A_{MAX}$ if and only if $\forall a_{l'} \in A \backslash a_l, a_l \not\subset a_{l'}$. With each set a_l is associated an incidence vector $A^l \in \{0,1\}^n$ $(A_i^l = 1$ iff $i \in a_l)$.

$$
\begin{array}{llll}
\min & z(M) = \displaystyle\sum_{a_l \in A_{MAX}} z_l & & \\
s.t. & \displaystyle\sum_{l \in A_{MAX}} A_i^l z_l \geq p_i & & \forall i \in \mathcal{A} \\
 & z_l \geq 0 & & \forall a_l \in A_{MAX}
\end{array}
$$

Due to the still very large number of variables (one for each undominated feasible subset) Mingozzi et al approximate the optimal value of this LP by computing heuristic solutions of the dual program, that can be transformed into the weighted node packing problem of a non-oriented graph $\tilde{G} = (\mathcal{A}, \tilde{E})$ where $(i, j) \in \tilde{G}$ if and only if a feasible subset exists, containing both activities i and j. It actually consists in finding in $\tilde{G}$ an independent set of activities of maximum total processing time.

7.4.3.2 Column generation techniques and further improvements.
In (Baar et al., 1998) and (Brucker and Knust, 2000), a new lower bound based on a destructive approach is presented (see Introduction). To detect infeasibility of trial value D, they first use constraint propagation techniques, including interval consistency, immediate selection, edge-finding and symetric triples (Brucker et al (1998)). If it is not sufficient to prove infeasibility, they try to prove that the solution of a LP-formulation called (BK) inspired from (M) (through a specific column generation algorithm) does not lead to a feasible schedule with makespan lower or equal to D. Note that the time window $[ES_i, LF_i]$ of each activity $i \in \mathcal{A}$ has been strengthened.
To present this LP-formulation, they first divide the time horizon $[0; D]$ into several contiguous subintervals: let $t_0 < t_1 < ... < t_L$ denote the ordered sequence of all different events ES_i and LF_i. For all $l \in \{1, \ldots, L\}$, γ_l denotes the total number of feasible subsets of $[t_{l-1}, t_l]$ and $a_{j,l}$ $(1 \leq j \leq \gamma_l)$ denotes all feasible subsets of the interval. They also associate with each set $a_{j,l}$ an incidence vector $A^{j,l} \in \{0, 1\}^n$ ($A_i^{j,l} = 1$ *iff* $i \in a_{j,l}$). We have one variable $z_{j,l}$ per feasible subset in an interval $[t_{l-1}, t_l]$. It denotes the number of time units where all activities in $a_{j,l}$ are processed simultaneously. Non-negative artificial variables u_l, $l \in \{1, \ldots, L\}$ are also introduced in order to turn the decision problem into an optimization problem. If precedence and non-preemption constraints are relaxed, it is easy to see that the scheduling problem is feasible if and only if the following linear program has the optimal value zero.

$$\begin{array}{lll}
\min z(BK) = & \sum_{t=1}^{L} u_l & \\
s.t. & \sum_{l=1}^{L} \sum_{j=1}^{\gamma_l} A_i^{j,l} z_{j,l} \geq p_i & \forall i \in \mathcal{A} \\
& \sum_{j=1}^{\gamma_l} z_{j,l} - u_l \leq t_{l+1} - t_l & \forall l \in \{1, \ldots, L\} \\
& z_{j,l} \geq 0 & \forall l \in \{1, \ldots, L\}, \forall j \in \{1, \ldots, \gamma_l\} \\
& u_l \geq 0 & \forall l \in \{1, \ldots, L\}
\end{array}$$

The pricing subproblem consists in scanning iteratively through the intervals $[t_{l-1}, t_l], 1 \leq l \leq L$ and searching for feasible subsets $a_{j,l}$ of executable activities in the current interval such that $\sum_{i=1}^{n} A_i^{jl} \theta_i - \varsigma_l > 0$, where θ_i and ς_l are the dual variables of the n first and the L next constraints of (BK), respectively. This involves the multidimensional knapsack-stable problem, that Brucker and Knust compute with a branch-and-bound procedure. This process generates several improving columns, and the solution of the restricted master problem is computed, before restarting it. If no improving column is found in each interval, then the optimal solution is found.
Baptiste and Demassey introduce in (Baptiste and Demassey, 2004) several cuts to be added to this linear formulation (the pricing subproblem is still the same, although they model it through a mixed integer program solved by CPLEX). As a matter of fact, this interval decomposition of $[0; D]$ allows some reflexions on the duration time an activity will be executed on each interval, because of *energetic*, non-preemptive or precedence reasons.
The "energetic reasoning" (Erschler et al., 1991) (Lopez et al., 1992), described in section 7.2.2.4, gives the minimum duration time of an activity i on an interval $[t_l; t_{l'}], l, l' \in \{1, \ldots, L\}, l < l'$: it is the minimum of:

1 the length of the interval;

2 the number of time units during which i is processed after time t_l if i is left-shifted, i.e. scheduled as soon as possible;

3 the number of time units during which i is processed before time $t_{l'}$ if i is right-shifted, i.e. scheduled as late as possible.

Then the non-preemption constraints can give an upper bound of the duration of an activity on several non-overlapping time intervals. For example, let i be an activity with $ES_i = 1$, $LF_i = 12$, and $p_i = 5$ and consider two intervals $[1, 4]$ and $[9, 11]$. In a non-preemptive schedule, i cannot overlap with both

$[1,4]$ and $[9,11]$ since $9-4 \geq p_i$. Hence, i is processed during at most 3 time units in $[1,4] \cup [9,11]$.

Finally, Baptiste and Demassey use this interval decomposition (a sort of discretization) to deduce some consequences of the precedence constraints between activities. This involves the mid-points $mid_{i,l}$ of the activity i on each interval $[t_{l-1}, t_l]$, that can be related to the $z_{j,l}$ variables, where the feasible subsets $a_{j,l}$ contain activity i. As the global mid-point mid_i of activity i is the weighted average of all the mid-points of i, we have for each $(i,j) \in E$, $mid_j - mid_i \geq \frac{p_j - p_i}{2}$.

These three remarks are modeled as inequalities involving the current variables of (BK). They can be used as cutting planes in the LP-formulation, and will improve the first results of (Brucker and Knust, 2000) which were known as the best lower bounds of the RCPSP. Note first that Baptiste and Demassey perform more constraint propagation: they solve initially a mixed integer linear program where single redundant machines are added. Such a machine contains activities that cannot overlap in time, through precedence or resource constraints. Edge finding and specific branching schemes are designed and applied to this redundant machines model, in order to strengthen even more the disjunctions between activities.

The impact of these single constraint propagation techniques is important, insofar as 12 of the 119 still open instances of (Kolisch (1996)) with 60 activities are closed in an average CPU-time of 1 s, and the solution of the linear relaxation (BK) with this preprocessing step improves 34 other lower bounds. The new cutting planes presented above improve then 21 lower bounds of these instances, closing 4 of them.

7.4.3.3 Feasible configuration and Redundant Resources. In (Carlier and Néron, 2003), the authors present a linear programming scheme for computing a lower bound for the RCPSP. It uses a set of K LLBs: $\{LLB_1, \ldots, LLB_K\}$ (see section 7.2.2.3) and a segmentation of the time horizon into successive intervals. These LLBs are those associated with the initial resources, redundant resources and paths in the conjunctive graph. All the LLBs associated with the redundant resources are taken into account. Indeed, if $(i_1, i_2, \ldots, i_r)$ is such a path, the LLB given by $p_{i_1} + p_{i_2} + \ldots + p_{i_r}$ is valid. The LLBs corresponding to the 10 longest paths are used in the tests.

In the linear program below, the time horizon $[0, t_{L+1}]$ is divided into L intervals: $[t_1 = 0, t_2]$, $[t_2, t_3]$, $\ldots$, $[t_L, t_{L+1}]$. t_{L+1} is the makespan of the schedule. $\mu_{i,l}$ is that part of activity i which is processed in $[t_l, t_{l+1}]$:

$$\begin{array}{lll}
\min & t_{L+1} & \\
s.t. & \sum_{l=1}^{L} \mu_{i,l} = p_i & \forall i \in \mathcal{A}, \\
& \mu_{i,l} \leq t_{l+1} - t_l & \forall l \in \{1, \ldots L\}, \forall i \in \mathcal{A}, \\
& LLB_h(\mu_{1,l}, \mu_{2,l}, \ldots \mu_{n,l}) \leq (t_{l+1} - t_l) & \forall l \in \{1, \ldots, L\}, \forall h \in \{1 \ldots, K\} \\
& \mu_{i,l} \geq 0 & \forall l \in \{1, \ldots, L\}, \forall i \in \mathcal{A} \\
& t_1 = 0 \leq t_2 \leq t_3 \leq \ldots \leq t_{L+1} &
\end{array}$$

For these tests, authors have considered a set of time-points: $T = \{ES_i, LF_i, \forall i \in \mathcal{A}\} = \{t_1, t_2, \ldots, t_{L+1}\}$. They assume that T is sorted in a non-decreasing order and that all time-points are different. So we add the two following linear constraints:

$$\forall l \in \{1, \ldots, L\}, \forall i \in \mathcal{A}, \text{ if } LF_i \leq t_l \text{ then } \mu_{i,l} = 0$$
$$\forall l \in \{1, \ldots, L\}, \forall i \in \mathcal{A}, \text{ if } ES_i \geq t_{l+1} \text{ then } \mu_{i,l} = 0$$

This linear program has some similarities with that presented by (Brucker and Knust, 2000), including the segmentation of the time horizon. Brucker and Knust generate subsets of activities that cannot be processed simultaneously because of resource and precedence constraints. Their model appears slightly more efficient than the one presented above for a small number of activities, but its drawback is that it is based on column generation, so its complexity may become exponential. This method, on the other hand, is polynomial and therefore scalable.

Experiments prove the efficiency of this bounding method. It is benchmarked on the 480 KSD instances with 120 activities (Kolisch (1996)) using the lower and upper bound reported on the PSPLIB web page as of October, 1st, 2004.

- the average gap between the lower bound and the best reported one is very small (smaller than 0.34%)

- the average and maximum computation times are still reasonable even for large instances (average time: 14.4s max time: 99.5s). Moreover this time can be decreased if only a subset of LLB that are used to get these results is kept: our bound is scalable.

- the redundant resources are useful for both computing energetic satisfiability tests and energetic time bound adjustments: the gap between the best known lower bound and the destructive energetic lower bound decreases from 4.4 % to 0.75% if the redundant resources are used.

- the number of instances where the bound is worse than the best known bound decreases significantly from 156 to 145, if the linear programming formulation based on MRF is used.

7.5 Conclusion and further research directions

In this state-of-the-art survey, we have arbitrarily divided the lower bounds proposed in the literature for the RCPSP according to the way the resource constraints are considered.

Another classification has to be underlined. On one hand there are fast computable lower bounds that do not give the best relaxations but that can be included inside branch-and-bound exact methods. These are longest-path based lower bounds, one machine and identical parallel machine relaxations, constraint propagation techniques such as edge-finding and energetic reasoning, the node-packing bound of Mingozzi et al On the other hand, there are more time-consuming methods that give better relaxations mainly based on the cooperation between linear and constraint programming and that can be applied only at the root node.

An important issue for future research is to close the gap between these two categories. This can be done by developing branch-and-cut or branch-and-price methods. Some interesting alternative recent work have been carried in this direction : lagrangean relaxation which significantly accelerates the LP resolution, Linear programming based on MRF, analysis of the forbidden set structure.

References

Alvarez-Valdés, R. and Tamarit, J. M. (1993). The project scheduling polyhedron: dimension, facets and lifting theorems, *European Journal of Operational Research*, 67:204–220.

Applegate, D. and Cook, W. (1991). A computational study of job-shop scheduling, *ORSA Journal on Computing*, 3(2):149–156.

Baar, T., Brucker, P., and Knust, S. (1998). Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. in: *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. Osman and C. Roucairol, eds, Kluwer, pp. 1–18.

Balas, E. (1970). Project scheduling with resource constraints, in: *Applications of Mathematical Programming Techniques*, Beale, E.M.L., ed, American Elsevier.

Baptiste, Ph. and Demassey, S. (2004). Tight LP bounds for resource constrained project scheduling, *OR Spectrum*, 26:251–262.

Baptiste, Ph. and Le Pape, C. (2000). Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems, *Constraints*, 5:119–139.

Baptiste, Ph., Le Pape, C., and Nuijten, W. (1999). Satisfiability tests and time-bound adjustments for cumulative scheduling problems, *Annals of Operations Research*, 92:305–333.

Bartusch, M., Möhring, R. H., and Radermacher, F. J. (1988). Scheduling project networks with resource constraints time windows, *Annals of Operations Research*, 16:201–240.

Brucker, P. (2002). *Scheduling algorithms*, Springer Verlag, Berlin.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling problem: Notation, classification, models and methods, *European Journal of Operational Research*, 112(1):3–41.

Brucker, P. and Knust, S. (2000). A linear programming and constraint propagation-based lower bound for the RCPSP, *European Journal of Operational Research*, 127:355–362.

Brucker, P., Knust, S., Schoo, A., and Thiele, O. (1998). A branch and bound algorithm for the resource-constrained project scheduling problem, *European Journal of Operational Research*, 107:272–288.

Carlier, J. and Latapie, B. (1991). Une méthode arborescente pour résoudre les problèmes cumulatifs, *RAIRO-Recherche Opérationnelle*, 25(3):311–340.

Carlier, J. and Néron, E. (2000). A new LP based lower bound for the cumulative scheduling problem, *European Journal of Operational Research*, 127(2):363–382.

Carlier, J. and Néron, E. (2003). On linear lower bounds for the resource constrained project scheduling problem, *European Journal of Operational Research*, 149:314–324.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem, *Management Science*, 35:164–176.

Carlier, J. and Pinson, E. (1990). A practical use of Jackson's preemptive schedule for solving the job-shop problem, *Annals of Operations Research*, 26:269–287.

Carlier, J. and Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem, *European Journal of Operational Research*, 78:146–161.

Carlier, J. and Pinson, E. (1998). Jackson's pseudo preemptive schedule for the $pm/r_i, q_i/c_{max}$ scheduling problem, *Annals of Operations Research*, 83:41–48.

Carlier, J. and Pinson, E. (2004). Jackson's pseudo preemptive schedule and cumulative scheduling problems, *Discrete Applied Mathematics*, 145:80–94.

Caseau, Y. and Laburthe, F. (1996). Cumulative scheduling with task intervals, in: *Proceedings of the Joint International Conference and Symposium on*

Logic Programming, JCPSLP'96, Maher, M., ed, The MIT Press, Cambridge, MA, pp. 363–377.

Christofides, N., Alvarez-Valdés, R., and Tamarit, J. M. (1987). Project scheduling with resource constraints: a branch and bound approach, *European Journal of Operational Research*, 29(3):262–273.

Demassey, S., Artigues, Ch., and Michelon, Ph. (2002). A hybrid constraint propagation-cutting plane algorithm for the RCPSP, in: *Proceedings of the 4th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR'02*, pp. 321–331.

Demassey, S., Artigues, Ch., and Michelon, Ph. (2005). Constraint-propagation-based cutting planes: an application to the resource-constrained project-scheduling problem, *INFORMS Journal on Computing*, 17(1).

Demeulemeester, E. and Herroelen, W. (2002). *Project scheduling. A research handbook*, Kluwer Academic Publishers.

Dyer, M. E. and Wolsey, L. A. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program, *Discrete Applied Mathematics*, 26:255–270.

Erschler, J., Lopez, P., and Thuriot, C. (1991). Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement, *Revue d'Intelligence Artificielle*, 5:7–32.

Fekete, S. P. and Schepers, J. (1998). New classes of lower bounds for bin packing problems, *Lecture Notes in Computer Science*, 1412:257–270.

Garaix, T., Artigues, Ch., and Demassey, S. (2005). Bornes inférieures et supérieures pour le RCPSP, in: *Proceedings of 6ième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF 2005*, Tours, France, pp. 219–240,

Guéret, Ch. and Prins, Ch. (1999). A new lower bound for the open-shop problem, *Annals of Operations Research*, 92:165–183.

Haouari, M. and Gharbi, A. (2003). An improved max-flow-based lower bound for minimizing maximum lateness on identical parallel machines, *Operations Research Letters*, 31:49–52.

Horn, W.A. (1974). Some simple scheduling algorithms, *Naval Research Logistic Quarterly*, 21:177–185.

Johnson, D. S., Demers, A. J., Ullman, J. D., Garey M. R. and Graham, R. L. (1974). Worst case performance bounds for simple one-dimensional packing algorithms, *SIAM Journal of Computing*, 3:299–325.

Klein, R. and Scholl, A. (1999). Computing lower bound by destructive improvement: An application to resource-constrained project scheduling, *European Journal of Operational Research*, 112:322–346.

Kolisch, R. and Sprecher, A. (1997). PSPLIB - a project scheduling problem library, *European Journal of Operational Research*, 96:205–216.

Laborie, Ph. (2003). Algorithms for propagation of resource constraints in AI planning and scheduling: Existing approaches and new results, *Artificial Intelligence*, 143:151–188.

Laborie, Ph. (2005). Complete MCS-based search: Application to resource constrained project scheduling, in: *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-05.*

Laborie, Ph. and Ghallab, M. (1995). Planning with sharable resource constraints, in: *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-95.*

Le Pape, C. (1994). Implementation of resource constraints in ILOG SCHEDULE: A library for the development of constraint-based scheduling systems, *Intelligent Systems Engineering*, 3(2):55–66.

Lopez, P., Erschler, J., and Esquirol, P. (1992). Ordonnancement de tâches sous contraintes: une approche énergétique, *R.A.I.R.O. APII*, 26(5–6):453–481.

Martello, S. and Toth, P. (1990). Lower bound and reduction procedure for the bin-packing problem, *Discrete Applied Mathematics*, 28:59–70.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact algorithm for the multiple resource-constrained project scheduling problem based on a new mathematical formulation, *Management Science*, 44:714–729.

Möhring, R. H., Schultz, A., Stork, F., and Uetz, M. (2003). Solving project scheduling problems by minimum cut computations, *Management Science*, 49:330–350.

Nuijten, W. (1994). *Time and resource constrained scheduling: A constraint satisfaction approach.* PhD thesis, Eindhoven University of Technology.

Pritsker, A. A., Watters, L. J., and Wolfe, P. M. (1969). Multi-project scheduling with limited resources: a zero-one programming approach, *Management Science*, 16:93–108.

Sankaran, J. K., Bricker, D. L., and Juang, S.-H. (1999). A strong fractionnal cutting-plane algorithm for resource-constrained project scheduling, *International Journal of Industrial Engineering: Applications and Practice*, 6(2):99–111.

Schwindt, Ch. (2005). *Resource Allocation in Project Management*, GOR-Publications.

Tercinet, F., Lenté, Ch., and Néron, E. (2004). Mixed satisfiability tests for multiprocessor scheduling with release dates and deadlines, *Operations Research Letters*, 32(11):326–330.

Vilain, M. and Kautz, H. (1986). Constraint propagation algorithms for temporal reasoning, in: *Proceedings of Fifth National Conference on Artificial Intelligence*, pp. 377–382.

Chapter 8

JUSTIFICATION TECHNIQUE GENERALIZATIONS

Vicente Valls[1], Francisco Ballestín[2], Sacramento Quintanilla [3]

[1]*Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot, Valencia, Spain.*

Vicente.Valls@uv.es

[2]*Dpto. de Estadística e Investigación Operativa, Facultad de Ciencias Económicas y Empresariales, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006, Pamplona, Spain.*

Francisco.Ballestin@unavarra.es

[3]*Dpto. de Economía Financiera y Matemática, Facultad de Económicas y Empresariales, Universitat de Valencia, Avda. de los Naranjos, s/n, Edificio Departamental Oriental, Valencia, Spain.*

Maria.Quintanilla@uv.es

Abstract The justification technique was introduced various decades ago for the resource-constrained project scheduling problem, although it has rarely been used with the problem. Justification is a simple and quick technique which when applied to schedules produces a new schedule that is, at most, as long as the original schedule - and often shorter. A recent article (Valls et al, 2005), showed that incorporating justification in heuristic algorithms can produce a substancial improvement in the results obtained. These results have motivated us to generalise this technique in order to study it in greater depth. This paper proposes distinct forms and generalisations for the justification technique and studies the relation existing among sets of obtainable schedules. The obtained results show that the proposed generalisations are worthwhile. Several computational tests have been performed to ascertain the impact of the generalisations on algorithmic efficiency.

Keywords: resource constrained project scheduling, justification, heuristics

8.1 Introduction

Wiest introduced the concepts of left-justified and right-justified schedules in 1964 with the objective of later defining the concepts of activity slack and critical schedule in the case of the resource-constraint project scheduling problem (RCPSP). Nevertheless, the applicability of justification is much wider; it can be used to significantly improve schedules generated by other methods. Valls et al (2005) show that justification - more specifically, double justification - is a simple and quick technique that can be incorporated in very diverse algorithms for the RCPSP and generates significant improvements in the quality of the schedules generated. They perform a set of experimental computations to compare the quality of 22 diverse algorithms (quick and simple, medium quality, state-of-the-art) with and without double justification. As test instances, they use the standard j120 set generated using ProGen (Kolisch et al (1995)) for the RCPSP. The number of generated schedules is limited to 5000 for all algorithms. In all the cases, the inclusion of double justification significantly improves the quality of the original algorithms. In total, some 14 of the new algorithms produce significantly better solutions than the state-of-the-art algorithms that do not use justification. To be more specific, the percentage deviation from the lower bound critical path for the best of the new algorithms with double justification, improves by more than 2 percent on the best quality state-of-the-art algorithms that do not use double justification. In addition, the improvement in quality is always obtained without an increase in computing time - and in some cases with a significant reduction in computing time. Based on the obtained results, the authors conclude that double justification should always be considered when designing an algorithm for the RCPSP.

Valls et al (2005) use a specific form of schedule justification that differs from that proposed by Wiest (1964) and in this paper we have termed this form of justification as *justification by extremes*. Although we give a formal definition in Section 16.3, for the purposes of the introduction we can say that right (left) justifying a schedule consists in moving the activities to the right (left) as much as possible in decreasing (increasing) order of their finishing (starting) times. Justification by extremes has been used by Li and Willis (1992), Tormos and Lova (2001), and Valls et al (2003a), Valls et al (2003b), Valls et al (2004), Valls et al (2005).

The good results obtained using justification by extremes open a new area of research interest. It seems worthwhile investigating other more general forms of justifying a schedule that could lead to further improvements. Notice that the result of right (left) justifying by extremes of a schedule S may not be unique if there is a tie in the starting (finishing) time of the activities. It is possible that different tie-breaking rules produce different schedules. Given a schedule S, we discuss the set of schedules $XRJ\ (S)$ $(XLJ\ (S))$ that may be obtained by right (left) justifying S by extremes. In other words, $XRJ\ (S)$ $(XLJ\ (S))$ is the set of reachable schedules from S using right (left) justification by extremes.

One way of generalising the technique of justification by extremes consists in, firstly, extending the set of reachable schedules $XRJ\ (S)$ $(XLJ\ (S))$ with a new set of schedules $RJ\ (S)\ (LJ\ (S))$ in such a way that their elements maintain the 'essential' properties of the schedules obtained by right (left) justification by extremes. Later, procedures must be designed that generalise the justification by extremes technique and can reach all of the $RJ\ (S)\ (LJ\ (S))$ set - or at least parts of it.

The rest of the paper is organised as follows: Section 16.2 introduces a set of previous definitions necessary for the development of the paper. Section 16.3 defines the new reachable sets of schedules $RJ\ (S)\ (LJ\ (S))$ and presents two strict generalisations of justification by extremes: justification by eligibles and general justification. It also studies the inclusion relations - mutually and with the sets of $RJ\ (S)$ and $LJ\ (S)$ - of schedules reachable by each of the three techniques. As schedules exist where these inclusions are strict, Section 16.4 proposes a new generalisation, which we will term shift, and also shows that any schedule of $RJ\ (S)\ (LJ\ (S))$ can be obtained from S by right (left) shift. Section 16.5 shows and comments on the results of the computational tests. Finally, the paper ends in Section 15.6 with some comments and concluding remarks.

8.2 Definitions

The resource-constrained project scheduling problem (RCPSP) may be stated as follows: A project consists of a set of n activities numbered 1 to n, where each activity has to be processed without interruption to complete the project. The dummy activities 1 and n represent the beginning and end of the project. The duration of an activity j is denoted by d_j where $d_1 = d_n = 0$. There are K renewable resource types. The availability of each resource type k in each time period is R_k units, $k = 1, \dots, K$. Each activity j requires r_{jk} units of resource k during each period of its duration where $r_{1k} = r_{nk} = 0$, $k = 1, \dots, K$. All parameters are assumed to be non-negative integer valued. There are precedence relations of the finish-start type with a zero parameter value (i.e., FS = 0) defined between the activities. In other words, activity i precedes activity j - if j cannot start until i has been completed. The structure of a project can be represented by an activity-on-node network $G = (V, A)$, where V is the set of activities and A is the set of precedence relationships. $S_j\ (P_j)$ is the set of successors (predecessors) of activity j. It is assumed that $1 \in P_j, j = 2, \dots, n$, and $n \in S_j\ , j = 1, \dots, n-1$. A schedule S is a set of starting times $(s_1, s_2, ..., s_n)$, $s_i \geq 0$, where the precedence and resource constraints are satisfied. The length of S is defined as $L(S) = s_n - s_1$. The objective of the RCPSP is to find a schedule S where the schedule duration $T(S) = s_n$ is minimised.

Note that if S is an optimal schedule then $s_1 = 0$. Given a schedule $S = (s_1, s_2, ..., s_n)$ with $s_1 > 0$ then the schedule $S' = (0, s_2 - s_1, \dots, s_n - s_1)$ is such that $s'_1 = 0$, has the same length as S, and a duration of $T(S') = s'_n = s_n - s_1$. Therefore, minimising the duration is equivalent to minimising the length.

8.3 Generalizing the justification by extremes

This section is devoted to the generalisation of the justification by extremes technique. It is divided into two subsections. In the first, the new reachable sets are defined. In the second subsection, we propose and study two generalisations of the justification by extremes procedure: justification by eligibles and general justification.

8.3.1 New reachable sets

Let $S = (s_1, s_2, \ldots, s_n)$ be a given schedule.

Justify an activity $i \neq n$ to the right in S consists in scheduling the activity as late as possible without violating the precedence relations or resource restrictions and leaving the remaining starting times fixed.

Justify an activity $i \neq 1$ to the left in S consists in scheduling the activity as soon as possible without violating the precedence relations or resource restrictions and leaving the remaining starting times fixed.

A schedule where no activity can be justified to the left is termed *left justified* or *left active*. Note that in this case $s_1 = 0$. In the same way, a schedule where no activity can be justified to the right is termed as *right justified* or *right active*.

A schedule S' is a *schedule justified to the right* of S if this is justified to the right, $s'_n = T(S)$ and $\forall i \; s'_i \geq s_i$. If $s'_1 > s_1$, S' is $s'_1 - s_1$ units shorter than S.

A schedule S' is a *schedule justified to the left* of S if this is justified to the left, $s'_1 = 0$ and $\forall i \; s'_i \leq s_i$. If $s'_n < T(S)$, S' is $T(S) - s'_n$ units shorter than S.

We denote by $RJ\ (S)$ $(LJ\ (S))$ the set of right (left) justified schedules of S.

Below, we propose two examples that show two extreme cases. In the first, by justifying a schedule to the right we obtain the optimal solution. In the second, we show a schedule that is justified to the right and left and yet, in contrast, is not optimal.

Example 1. - Let us look at the project in Figure 16.1. S is an active schedule. If we justify in S the activities 1 and 2 to the right we obtain S'= $(0, 2, 1, 2, 3)$, a schedule justified to the right with $s'_1 = 1$ and length 2. The schedule $S'' = (0, 2-1, 1-1, 2-1, 3-1)$ is justified to the right and left and is optimal.

Example 2. - Let us study the project shown in Figure 16.2. No S activity can be justified to the right or left, therefore, the schedule is justified to the right and left. Nevertheless, it is not optimal as the optimal length is 5, as shown in S'. This example illustrates that the intersection between the set of right

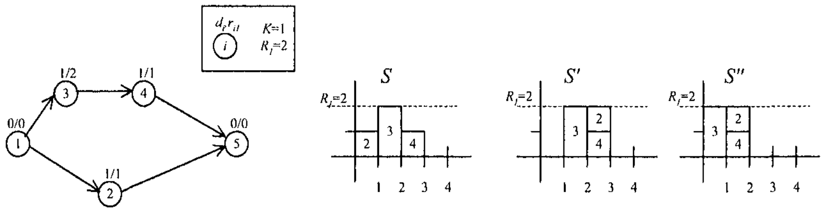


Figure 8.1. Schedule justified once to the right and is optimal.

justified schedules and the set of left-justified schedules $(RJ\ (S) \cap LJ\ (S))$ of a given schedule may not be empty and may not contain any optimal schedule - even if S is an active schedule.

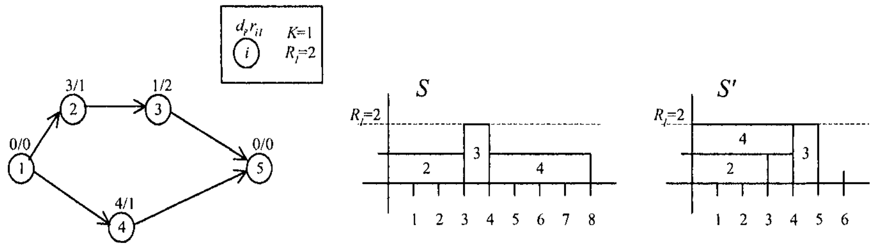


Figure 8.2. Schedule S is right and left justified yet it is not optimal. Schedule S' is optimal.

8.3.2 Two generalisations

Right justification by extremes of a schedule S consists in obtaining a schedule S' justified to the right, justifying all the activities in decreasing order of finishing times. The jth justification is carried out on a schedule obtained after undertaking $j - 1$ earlier justifications.

We will term $XRJ\ (S)$ the set of schedules obtainable using right justification by extremes of a schedule S.

Note that if there are no two activities that finalise at the same moment, the set $XRJ\ (S)$ is formed by a single schedule.

Analogously, we can define *left justification by extremes* of a schedule, justifying the activities in growing order of the starting times and the set XLJ (S).

Right justification by eligibles of a schedule S consists in obtaining a schedule right justified S', applying a finite number of right justifications of the activities in S in such a way that the following condition is satisfied: an activity is eligible for justification if its successors and any activity that finishes after any successors are right justified. This condition implies that no successor of an eligible activity can be justified to the right - independently of the right justifications later undertaken. The jth justification is carried out on a schedule obtained when undertaking the $j-1$ previous justifications.

We term ERJ (S) the set of schedules obtainable by the right justification by eligibles of a schedule S.

Analogously, *left justification by eligibles* of a schedule S and the set ELJ (S) can be defined. In this case, an activity is eligible for justification if its predecessors and any activity that starts before any predecessors are left justified.

General right justification of a schedule S consists in obtaining a schedule S' justified to the right, applying a fixed number of right justifications of the activities in S. The jth justification is carried out on a schedule that reflects the $j-1$ previous justifications.

We term GRJ (S) the set of schedules obtainable by general right justification of a schedule S.

Analogously, we can define *general left justification* of a schedule S and set GLJ (S).

Given that justification by extremes is a particular case of justification by eligibles and general justification, the following proposition is obvious.

PROPOSITION 8.1 *All schedules S fulfil:*

$XRJ\ (S) \subseteq ERJ\ (S) \subseteq GRJ\ (S) \subseteq RJ\ (S)$

$\min\{L(S')/S' \in XRJ\ (S)\} \geq \min\{L\ (S')/S' \in ERJ\ (S)\}$

$\geq \min\{L\ (S')/S' \in GRJ(S)\} \geq \min\{L(S')\ /S' \in RJ\ (S)\ \}$

The following result shows that justification by eligibles and general justification are strict generalisations of justification by extremes. In particular, these generalisations enable us to obtain, at least sometimes, better quality schedules than could be obtained using justification by extremes.

PROPOSITION 8.2 *For each inclusion and each inequality in proposition 8.1, there is a schedule where the inclusion and inequality are strict.* □

The proof of the proposition can be divided into the following parts.

Part 1 - $\exists S/XRJ\ (S) \subset ERJ\ (S)$ and $\min\{L(S')/S' \in XRJ\ (S)\} >$ $\min\{L(S')/S' \in ERJ\ (S)\}$.

Figure 16.3 shows a project and three schedules: S, S' and S''. There are only two ways to justify the active schedule S by extremes, given that the only activities that finish at the same time are 3 and 5. Both ways lead to schedule S', of the same length as S. It is obvious that if we left justify S' by extremes then the schedule S will be obtained, and therefore justifying by extremes will not improve S - even when applied repeatedly. Nevertheless, justification by eligibles enables activity 2 to be justified first and this activity can be selected at the first iteration. In this way S'' is obtained, which is right justified and has a length 3 units shorter than S.

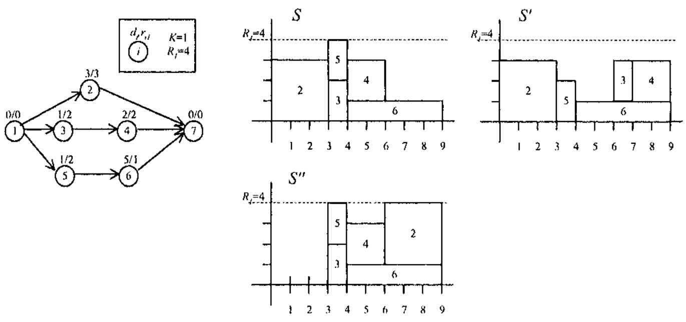


Figure 8.3. Example for part 1.

Part 2 - $\exists S/ERJ\ (S) \subset GRJ\ (S)$ and $\min\{L(S')/S' \in ERJ\ (S)\ \} > \min\{L(S')/S' \in GRJ\ (S)\ \}$.

Figure 16.4 shows a project and three schedules S, S' and S''. We can see that the only solution that can be obtained from active schedule S by using justification by eligibles is S''. The activities 11, 9, 8, 7, 5 and 3 cannot be justified to the right even though other activities are justified to the right. After fixing the fictitious activity 13 in $T(S)$ = 9, the eligible activities are 12 and 10. Activity 12 cannot be moved to the right, so activity 10 is justified to the right and subsequently scheduled at [8, 9]. The eligible activities are now 12 and 6, the first of which is justified to the right. We must now choose activity 6 and after justification it is scheduled at [7, 8]. The eligible activities are now 12 and 2. Both can be justified to the right and so two possibilities exist. Let us assume that we justify activity 2 first. In this case, activity 2 is scheduled at [1, 2], and therefore the only eligible activity is 12, which will be scheduled at [4, 7] when justified. The only remaining eligible activity is then 4, which is already justified to the right and the resulting schedule is S''. If on the other hand, we initially choose activity 12 it would be scheduled at [4, 7] and the eligible activities are 2 and 4. The last activity cannot be moved to the right and so we justify activity 2 to [1, 2]. Now just one activity is eligible and it cannot be justified, and so we also arrive at S''. This shows that S cannot be improved by using justification by eligibles. We will now see that it is possible to improve S by using general justification. To do this we must first justify activity 6, which will then be scheduled at [6, 7]. Then we justify activity 10 until [8, 9], and then 12 until [5, 8]. This will leave a space in [3, 5] sufficiently large for the justification of activity 4. Justifying activity 2 to [1, 2] creates a schedule S', which is justified to the right and whose length is 1 unit smaller than S.

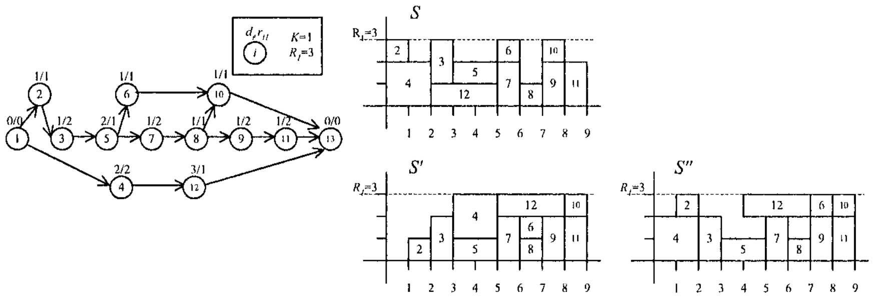


Figure 8.4. Example for part 2.

Part 3 - $\exists S/GRJ\ (S) \subset RJ\ (S)$ and $\min\{L(S')/S' \in GRJ\ (S)\ \} > \min\{L(S')/S' \in RJ\ (S)\ \}$.

Figure 16.5 shows a project and three schedules S, S' and S''. We are going to show that $GRJ\ (S) = \{S''\}$, S'' being the same length as S. Activities 11, 10, 9, 8, 7, 5 and 3 cannot be right justified even though other activities were justified to the right. At the beginning, only activities 6 and 2 could be moved to the right. Activity 3 is the successor of activity 2 and activity 3 cannot be moved to the right, so then activity 2 can only be moved to the interval [1, 2]. Once activity 2 is justified, the only unjustified activity is 6, and so we must select this activity and schedule it at [7, 8]. Once done, activity 12 remains the only activity that has changed from being justified to the right to not being justified - and so we select this activity and justify it until [4, 7]. This action leads to S" which is right justified and the only schedule obtainable using general justification. Nevertheless, the schedule S' is justified to the right of S, and its length is one unit shorter than that of S and S''.■

Evidently, in the two earlier propositions, as well as the following section, right justification can be substituted for left justification. This can be shown by considering the reverse network - that is, the network that is obtained by simply reversing the precedence relations of the original project - and applying the earlier results to this network.

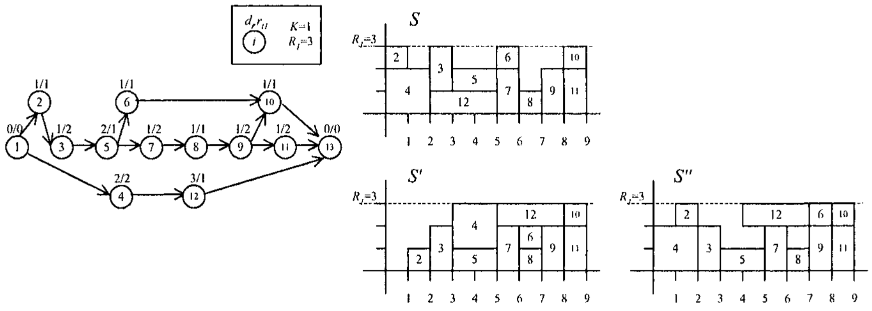


Figure 8.5. Example for part 3.

Notice that none of the three justifications state the order in which activities are justified. Different orders may result in different justified schedules.

It is also interesting to note that none of the three justifications can reduce the quality of a schedule.

It is also worth noting that justification by extremes means a maximum of n-1 justifications of activities. This is not so with general justification and justification by eligibles - because in both cases an activity can be justified more than once.

Also note that general justification is unable to completely generate the sets $RJ(S)$ and $LJ(S)$ in all cases.

8.4 Shift

Justifying an activity in a schedule to the right or left means moving it to the right or left as much as possible. Nevertheless, an activity can be moved in one of the two directions without reaching the limit. In this section we will formally define this new movement - which we will term *shift.* We will show that it is strictly more general than general justification and can generate all schedules in $RJ(S)$ and $LJ(S)$.

Shift to the right an activity $i \neq n$, with a movement $k \geq 0$ in a schedule S consists in obtaining the schedule S' with $\forall j \neq i\,(s'_j = s_j$ and $s'_i = s_i + k)$. Note that if S' does not satisfy the restrictions of the resources, then we consider that this activity cannot be shifted to the right in S with this movement.

Shift to the right a schedule S consists in obtaining a schedule S' justified to the right applying to S a finite number of activity shifts to the right with movements greater than 0.

We term $RT(S)$ the set of schedules obtainable using shifts to the right of S.

Analogously, we can define the *left shift of an activity* and the *left shift of a schedule* and the set $LT(S)$.

It is evident that shift generalises the concept of general justification. The question that needs answering is whether the shift and general justification of a schedule generate the same set of schedules; or if not, whether the length obtained by each technique is different. The following result answers these questions and shows how shift is better than justification.

PROPOSITION 8.3 *All schedules S fulfil:*

$GRJ\ (S) \subseteq RT(S)$ *and* $\min\{L(S')/S' \in GRJ\ (S)\} \geq \min\{L(S')/S' \in RT\ (S)\ \}$. *There is a schedule whereby the inclusion and previous inequality are strict.* □

The proof of the first inclusion and the first inequality is trivial. We use an example to show that the second affirmation is true.

Let us look at the project in Figure 16.5. The only schedule that can be obtained from S using general justification is S''. Therefore, the schedule S' cannot be obtained from S using general justification. We will see if it can be obtained by shift. This is trivial and it is enough to shift the activity 6 to [6, 7]. This creates a space in [5, 8] where the activity 12 can be moved to, and this enables activity 4 to be moved to [3, 5]. Finally, we move the activity 2 to [1, 2], and we arrive at S'. This shows that the second inclusion is strict. ■

This result shows that shift is not a trivial generalisation of general justification. However, the following result also shows that something else is true - namely, that by using shift it is possible to find all the right justified schedules of a given schedule. Therefore, additional generalisation is unnecessary.

PROPOSITION 8.4 $RT(S) = RJ\ (S)$ *is true for each schedule* S.□

The inclusion $RT(S) \subseteq RJ\ (S)$ is evident from the definitions. Let us have a look at the other inclusion.

Given an activity i and a schedule S, we term $s(i, S)$ $(f(i, S))$ the starting time (finishing time) of the activity i in the schedule S.

Let S be a schedule and $S' \in RJ\ (S)$. Let λ be the list of activities in decreasing order of their starting times in S'- the ties being resolved randomly. Notice that n is the first activity in λ. We are going to prove that S' can be obtained from S by shifting the activities in the order indicated by λ and with movement $s(i, S')$ - $s(i, S) \geq 0$ for each activity i.

Let S_p be the vector of activity starting times obtained after shifting the first p activities of λ in the order and with the movements previously indicated. We want to show that S_p is a schedule, $p = 1, \ldots, n$. We achieve this by induction over p.

As $s(n, S') = s(n, S)$, then $S_1 = S$ and S_1 is a schedule. Let us now assume, by induction hypothesis, that S_{p-1} is a schedule and let us show that S_p is a schedule.

Let i be the activity occupying position p in λ. S_p and S_{p-1} only differ in the i activity starting times: $\forall j \neq i\ s(j, S_{p-1}) = s(j, S_p)$, and $s(i, S_p) \geq s(i, S_{p-1})$. For this reason, S_p satisfies the precedence relations. Let us see if they also satisfy the resource restrictions. As S_{p-1} is a schedule, it is only necessary to prove that S_p meets the resource restrictions in the interval $[s(i, S_p), f(i, S_p)]$.

If $s(i, S_p) = s(i, S_{p-1})$, then $S_p = S_{p-1}$ and, so, S_p is a schedule.

Let us assume, on the contrary, that $s(i, S_p) > s(i, S_{p-1})$. If j is an activity so that $s(j, S_p) \geq s(i, S_p)$, then $s(j, S') \geq s(j, S_p) \geq s(i, S_p) = s(i, S')$. If $s(j, S') = s(i, S')$, so $s(j, S_p) = s(j, S')$. If $s(j, S') > s(i, S')$, then the activity j has been shifted before activity i and, so, cannot be shifted again: $s(j, S_p) = s(j, S')$.

So, in each time period of $[s(i, S_p), f(i, S_p)]$ S' schedules the same activities as S_p and, possibly more. As S' meets the resource restrictions then S_p also satisfies them. Therefore, S_p is also a schedule.■

Taking this proof into account, we can affirm that only n shifts are necessary in S to obtain any schedule justified to the right of S.

8.5 Computational tests

In the previous section we presented examples of various generalisations of justification by extremes that can generate better schedules than justification by extremes. Accordingly, research into ways of incorporating generalisations of justification by extremes in the design of heuristics seems promising. Nevertheless, it might be true that there are few instances in which improvements are possible, or that theoretically possible improvements could not be practically realised with algorithms. Therefore, it would seem worthwhile clarifying this question before studying in greater depth the development of algorithms based on generalisations of justification by extremes. This is the objective of the computational tests that we present in this section. The tests are centred on showing the potential of improvement of justification by eligibles compared to justification by extremes.

Before describing and commenting on the computational tests we will specify the justification procedures that we use. In justification by eligibles each schedule can be justified to the right in different ways, depending on the order in which the eligible activities are selected. In these algorithms we use three rules for selecting from within the set of eligible activities that activity that will be next justified. Each rule produces a distinct algorithm.

Rule 1. Randomly choose an activity.

Rule 2. *Right justification.* Choose an activity that finishes later if right justified. In case of a tie, choose the activity that finishes later in the current schedule. *Left justification.* Choose an activity that begins sooner if left justified. In case of a tie, choose the activity that begins sooner in the current schedule.

It is worth pointing out that when a schedule is justified following this rule, once an activity is justified it cannot be justified again. In this way, justification by eligibles with rule 2 can perform a maximum of $n - 1$ activity justifications.

Rule 3. *Right justification.* Firstly, the instant that each eligible activity terminates if right justified is calculated. Then the activity that finishes later in the current schedule is selected from the activities with the two longest times. *Left justification.* Firstly, the instant that each eligible activity begins if left justified is calculated. Then the activity that begins soonest in the current schedule is selected from the activities with the two shortest times.

It is worth noting that although the justification by eligibles with rules 2 and 3 are special cases of justification by eligibles, they are still non-superfluous generalisations of the justification by extremes - as shown by the example in Figure 16.3.

In justification by extremes, the following activity to be justified is randomly selected from the activities that finish (begin) at the same time.

As test instances, we have used the standard j120 set for the RCPSP generated using ProGen (Kolisch et al (1995)). The j120 set consists of 600 projects with four resource types and 120 activities. These instances were generated under a full factorial experimental design with the following three independent problem parameters: network complexity, resource factor, and resource strength. Details of these problem instances are given in Kolisch et al (1995) and Kolisch and Sprecher (1997). They are available in the Project Scheduling Problem Library (PSPLIB) (http://www.bwl.uni-kiel.de/Prod/psplib/) along with the optimum, or best known, values that have been obtained by various authors over the years. As of May 2005, an optimum solution for j120 instances had only been discovered in approximately 35% of instances. This shows the difficulty represented by the j120 set for current algorithms.

8.5.1 Justification of random schedules

The objective of this section is to compare the quality of the $ERJ\ (S)$ and $XRJ\ (S)$ $(ELJ\ (S)$ and $XLJ\ (S))$ sets. To do this, we have randomly generated 1666 active schedules for each instance of the j120 set, and for each instance we have applied ten times right justification by extremes and justification by eligibles with rule 1. To each right active schedule obtained after the first right justification by extremes we applied ten times left justification by extremes (by eligibles with rule 1). In this way, we generated approximately

Table 8.1. Justification by extremes and by eligibles on randomly generated schedules.

Algorithm	av_imp	max_imp	stand_dev
Random 1666 + XRJ	12.4359	-	-
Random 1666 + 10 XRJ	12.4357	12.8790	0.3348
Random 1666 + XRJ + XLJ	1.1462	-	-
Random 1666 + XRJ + 10 XLJ	1.1463	1.3820	0.1826
Random 1666 + ERJr1	7.5159	-	-
Random 1666 + 10 ERJr1	7.5152	10.5349	1.8667
Random 1666 + ERJr1 + ELJr1	1.3712	-	-
Random 1666 + ERJr1 + 10 ELJr1	1.3704	3.2460	1.0003

10 million (600x1666x10) schedules with each type of justification - resulting in approximately 40 million schedules.

The improvement over the initial solution was calculated each time a schedule was calculated: being the ratio between the length of a justified schedule minus the length of the initial schedule and the length of the initial schedule - expressed as a percentage.

Table 16.1 summarises the results. The first four rows refer to justification by extremes; the last four refer to justification by eligibles. The first column indicates the number, the type, the rule used when dealing with eligible justification and the side of the justification referred to in the results shown in the next three columns. Random 1666 + XRJ (Random 1666 + 10 XRJ) consists in generating the 1666 random schedules and right justifying by extremes only once (ten times). Random 1666 + ERJr1 + 10 ELJr1 consists in generating the 1666 random schedules, then right justifying them once by eligibles using rule 1 and then left justifying the obtained schedule ten times with rule 1. The remaining rows are treated similarly. The second column, *av_imp* consists of the average for all schedules and all instances of the percentage improvements obtained after the first justification in the Random 1666 + XRJ and the second justification Random 1666 + XRJ+ XLJ. In the cases of Random 1666 + 10 XRJ and 10-XLJ, the average is the average of the ten percentage improvements obtained after 10 justifications. The meaning of the 4 last rows is the same but with justification by eligibles. The average maximal (standard deviation) percentage improvement from the initial solution is reported in the third (fourth) column, labelled *max_imp (stand_dev).*

We can see that the average after the first right justification by eligibles, as well as after 10 justifications, is less than that of justification by extremes - and even the maximum after 10 justifications is less than the average after the first right justification by extremes. These results seem to indicate that the average quality of ERJ (S) schedules is considerably worse than that of XRJ (S). It can also be seen that the standard deviation after 10 justifications by

Table 8.2. Justification by eligibles with rule 2.

algorithm	av_imp	% improved
Random 1666 + ERJr2	12.76	98.66
Random 1666 + ERJr2 +ELJr2	1.73	61.16

extremes is much less than in the case of justification by eligibles - especially if the difference in averages is taken into account. This clearly implies a greater diversity in $ERJ\ (S)$ than in $XRJ\ (S)$.

Although we cannot compare the left justifications by extremes and by eligibles in absolute terms because they originate in solutions of differing quality, we can see that the average and maximum improvements are in all cases greater for justification by eligibles than for justification by extremes. This may be because the quality of the initial schedules for justification by extremes is better than the initial schedules for justification by eligibles. It can also be seen that there is a greater dispersion in the cases of justification by eligibles than in justification by extremes.

We have also applied justification by eligibles using rule 2. Firstly, we right justified the 1666 initial schedules and then we left justified the obtained schedules. The results are shown in Table 16.2. The terms have the same meanings as in Table 16.1. The percentage of solutions improved is reported in the third column, labelled % improved.

The best average obtained by right justifying with rule 2 is 12.76%, significantly greater than the maximum reached after 10 random justifications by eligibles (10.53%). This reaffirms the great diversity of $ERJ\ (S)$. Note that 98.66% of the schedules were improved by right justifying once; and that the improvements obtained are greater than those obtained with Random 1666 + XRJ and Random 1666 + 10 XRJ.

With these results in hand various conclusions can be made that can be applied to random active schedules. Justification by eligibles with random selection produces, in general, worse results than justification by extremes. Nevertheless, applying justification by eligibles with the appropriate rule can produce better solutions than justification by extremes.

8.6 A simple and efficient algorithm

To make the potential of justification more patent we have compared the results obtained in the previous section with the results obtained using state-of-the-art algorithms with the number of generated schedules limited to a maximum of 5000. The results obtained are shown in Table 16.3. The algorithm Random 5000 refers to the result of randomly generating 5000 active schedules. Note

Table 8.3. Random schedules with and without double justification.

algorithm	CP_dev
Random 5000	47.51
Random 1666 +XRJ+XLJ	38.36
Random 1666 +ERJr2+ELJr2	36.39

Table 8.4. State-of-the-art heuristic algorithms.

Author(s)	Algorithm type	CP_dev	DJ/Justification
Alcaraz (2001)	Genetic algorithm	36.57	No
Dorndorf et al (2000)	Truncated B&B	37.1	No
Hartmann (1998)	Activity list GA	36.74	No
Hartmann (2002)	Self-adapting GA	35.39	No
Kochetov and Stoylar (2003)	GA, TS, path relinking	33.36	Yes
Merkle et al (2002)	Ant colony optimisation	35.43	No
Tormos and Lova (2001)	Genetic algorithm	34.41	Yes
Valls et al (2005)	DJGA	33.24	Yes
Valls et al (2003b)	HGA	32.54	Yes

that the number of schedules generated by the last two algorithms is 4998 (=1666x3). The average percentage deviations of the solutions obtained by the different algorithms from the critical path lower bound - which is obtained by computing the length of a critical path in the resource relaxation of the problem, are shown in the second column.

Table 16.4 shows the results published by the authors of the different state-of-the-art algorithms when the number of generated schedules is limited to a maximum of 5000.

It is a clear sign of the power of justification that by simply randomly generating 1666 schedules and justifying them doubly, the resulting algorithm improves three of state-of-the-art algorithms that do not use justification.

8.6.1 Improving state-of-the-art algorithms

We have performed two other experiments consisting in incorporating the justification by eligibles into the two algorithms in Table 16.4 that obtain the best results.

Double justification consists in right justifying a schedule and subsequently justifying the obtained schedule to the left. Valls et al (2003b) incorporate double justification by extremes in the activity list GA of Hartmann (Hartmann (1998)). The new algorithm, DJGA, double justifies all schedules generated by the evolutionary process. The authors wrote their own implementation of the algorithm of Hartmann that is denoted by GA in this paper. The objective of the first experiment is to evaluate the impact of substituting in DJGA justification by extremes for justification by eligibles with rule 3 (GA+ERJr3+ELJr3).

The three algorithms were run with an upper limit of 5000 schedules. The size of the population in GA is 100 while the algorithms with justification use size 50. The average percentage deviations of the solutions obtained by the different algorithms from the critical path lower bound are 37.00, 33.24, and 32.78 for GA, DJGA, and GA+ERJr3+ELJr3, respectively.

GA+ERJr3+ELJr3 is an improvement on DJGA in quality, although the difference is not very great. It must be remembered that DJGA already produces very high quality solutions; in fact it is one of the best state-of-the-art algorithms. Nevertheless, we can state that the average run time of the current (not optimised) version of GA+ERJr3+ELJr3 is much longer than DJGA.

Justification by extremes is faster than justification by eligibles - at least in its current non-optimised implementation. It may be worthwhile combining both types of justification with the objective of obtaining a trade-off between schedule quality and run time. The following test offers evidence in favour of the above hypothesis. A recent paper (Valls et al (2003b)) shows that HGA outperforms all state-of-the-art algorithms for the RCPSP - at least on the standard j120 set. HGA is a hybrid genetic algorithm that double justifies by extremes all schedules generated by the evolutionary process. We have modified HGA by applying double justification by eligibles until 1000 schedules are generated and then double justification by extremes until 5000 schedules are generated. The modified HGA has a CP_dev value of 32.27 with an average run time of 4.74 seconds, whereas HGA shows a CP_dev value of 32.54 with an average run time of 2.03 seconds. Both algorithms ran on the same computer (PC 400 MHz). So, the modified HGA improves the best result known for 5000 schedules in a reasonable run time, establishing a new gap of 3 units between the best CP_dev values obtained by the algorithms with, and without, justification.

8.6.2 Further applicability of justification by eligibles

In the previous computational analysis, we have used three selection rules to test the performance of the justification by eligibles technique in the RCPSP. These rules have been designed to suit the characteristics and goal of the RCPSP. Similarly, the justification by eligibles can be tailored to other project scheduling problems by defining appropriate selection rules. In fact, we have successfully applied the justification by eligibles to two extensions of the RCPSP which appear when due dates in the activities are considered. In the first problem (TardinessRCPSP) the objective is total tardiness minimisation. In the second problem (DeadlineRCPSP) the due dates are strict (deadlines) and the objective is makespan minimisation. We have compared the performance on both problems of well-known RCPSP heuristics - priority rules, sampling procedures and metaheuristics - with new versions we have developed that take due dates into consideration. We have also analysed the effect of applying the justification by extremes and by eligibles to those algorithms. The computational results show that, in all cases, the justification by eligibles outperformed all other techniques tested. For further information, we refer to Ballestín et al (2006).

8.7 Summary and concluding remarks

In this paper we consider the usual concept of justification - justification by extremes - from a new perspective. We have broadened the usual justification concept by proposing that to justify a schedule S consists in generating a schedule in the $RJ\ (S)$ $((LJ\ (S))$ set. In this new context, justification by extremes is simply a basic technique for generating schedules of $RJ\ (S)$ $(LJ(S))$.

As justification by extremes does not always enable the generation of all the $RJ\ (S)$ $(LJ\ (S))$ schedules the door remains open to researching more general forms of justification. We have proposed three generalisations of justification by extremes, each of which strictly generalises either: justification by eligibles, general justification, and shift. However, other forms are possible. We have also seen how when using the same justification technique it is possible to obtain different algorithms by simply varying the rule for selecting the following activity to be justified. It is also worth noting that by using shift it is possible to find all the right (left) justified schedules of a given schedule. Therefore, additional generalisation is unnecessary.

The computational tests offer evidence to support the affirmation that the variability of the set of schedules reachable by justification by eligibles is considerably greater than the set of schedules reachable by justification by extremes. Therefore, it is not surprising that random justification by eligibles produces worse results than justification by extremes when both generate the same number of schedules. Nevertheless, justification by eligibles with rule 2 improves justification by extremes. So it would not be strange to wonder whether a more intelligent rule would not offer better quality solutions.

The obtained results also show that justification by eligibles improves randomly generated schedules - more than 98% of approximately one million random schedules - as well as those generated by quality algorithms. In fact, it is interesting to note that a procedure as simple as generating and justifying by eligibles random schedules is able to compete with state-of-the-art algorithms that do not use justification. As far as schedule quality is concerned, the modified HGA - with justification by eligibles - ranks first in the list of heuristics for the RCPSP in the case of 5000 schedules.

It is also worthwhile pointing out that the broader approach to the justification we propose enables the extension of the benefits of justification technique to project scheduling problems other than the RCPSP. Justification by extremes works on a schedule independently of the characteristics of the problem. However, justification by eligibles can be tailored to the goal of the optimisation at hand. An appropriate definition of the selection rule would favour, for example, moving first those activities with greater cash flow, or greater tardiness, or greater weight, depending on the objective function.

In our opinion, these results indicate that the study of new forms of justification is a promising research topic that could lead to further algorithmic advances for solving the RCPSP and other related project scheduling problems.

Acknowledgements

This research was partially supported by the Ministerio de Ciencia y Tecnología under contract TIC2002-02510 and the Agencia Valenciana de Ciencia y Tecnología, GRUPOS03/174.

References

Alcaraz, J., and Maroto, C. (2001). A robust genetic algorithm for resource allocation in project scheduling, *Annals of Operations Research*, 102:83–109.

Ballestín, F., Valls, V., and Quintanilla, S. (2006). Due Dates and RCPSP, in: *Perspectives in Modern Project Scheduling*, J. Józefowska and J. Weglarz, eds, Kluwer, pp. 79–104.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2000). A branch-and-bound algorithm for the resource-constrained project scheduling problem, *Mathematical Methods of Operations Research*, 52:413–439.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling, *Naval Research Logistics*, 45:733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints, *Naval Research Logistics*, 49(5):433–448.

Kochetov, Y., and Stolyar, A. (2003). Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem, in: *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies.*

Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems, *Management Science*, 41:1693–1703.

Kolisch, R., and Sprecher, A. (1997). PSPLIB - A project scheduling library, *European Journal of Operational Research*, 96:205–216.

Li, KY., and Willis, RJ. (1992). An iterative scheduling technique for resource-constrained project scheduling, *European Journal of Operational Research*, 56:370–379.

Merkle, D., Middendorf, M., and Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling, *IEEE Transaction on Evolutionary Computation*, 6:333–346.

Tormos, P., and Lova, A. (2001). A competitive heuristic solution technique for resource-constrained project scheduling, *Annals of Operations Research*, 102:65–81.

Valls, V., Quintanilla, S., and Ballestín, F. (2003a). Resource-constrained project scheduling: a critical activity reordering heuristic, *European Journal of Operational Research*, 149:282–301.

Valls, V., Ballestín, F., and Quintanilla, S. (2003b). A hybrid genetic algorithm for the RCPSP. Technical Report, Departamento de Estadística e Investigación Operativa, Universidad de Valencia.

Valls, V., Ballestín, F., and Quintanilla, S. (2004). A population-based approach to the resource-constrained project scheduling problem, *Annals of Operations Research*, 131:305–324.

Valls, V., Ballestín, F., and Quintanilla, S. (2005), Justification and RCPSP: a technique that pays, *European Journal of Operational Research*, 165:375–386.

Wiest, J D. (1964). Some properties of schedules for large projects with limited resources, *Operations Research*, 12:395–418.

Chapter 9

A METAHEURISTIC APPROACH TO THE RESOURCE CONSTRAINED PROJECT SCHEDULING WITH VARIABLE ACTIVITY DURATIONS AND CONVEX COST FUNCTIONS

Koji Nonobe
Department of Art and Technology, Faculty of Engineering,
Hosei University
3-7-2 Kajinocho, Koganei, Tokyo 184-8584, Japan
nonobe@k.hosei.ac.jp

Toshihide Ibaraki
Department of Informatics, School of Science and Technology,
Kwansei Gakuin University
2-1 Gakuen, Sanda, Hyogo 669-1337, Japan
ibaraki@ksc.kwansei.ac.jp

Abstract We introduce a generalized model of the resource constrained project scheduling problem (RCPSP). It features that (i) the duration of an activity is not constant, but can vary in a specified range, and (ii) the objective is to minimize a convex function of time-lag costs, where a time-lag cost is charged according to the difference between the start/completion times of activities. These features achieve the flexibility of the model. It is known that, in the RCPSP, resource constraints can be replaced by some precedence constraints appropriately defined between the activities that require a common scarce resource. If we remove resource constraints by precedence constraints, our problem can be formulated as the dual problem of a minimum cost flow problem, and thus can be solved efficiently. Exploiting this property, we design a heuristic algorithm based on local search. We conducted computational experiments with benchmark instances to minimize the weighted earliness-tardiness costs, as well as instances in which activity-crashing or relaxation of temporal constraints are allowed. These results indicate the usefulness of our generalized RCPSP model and the proposed algorithm.

Keywords: Resource constrained project scheduling, convex objective functions, variable activity durations, metaheuristics.

9.1 Introduction

As the resource constrained project scheduling problem (RCPSP) has a wide range of applications, its models, extensions and algorithms have been extensively studied (Brucker et al (1999), Demeulemeester and Herroelen (2002), Klein (2000), Herroelen et al (1998), Neumann et al (2002), Weglarz (1999)). Although the objective functions studied in many of these models are regular (e.g., project duration and tardiness), the importance of non-regular ones have also been recognized. Among non-regular objective functions, we concentrate on convex objective functions in this paper. A typical example of convex functions is the earliness-tardiness cost, and, in order to increase applicability, we also allow the *time-lag costs*, which are charged according to the differences between the start/completion times of activities. Then, we consider the RCPSP to minimize a convex function of time-lag costs (RCPSP/conv). Another feature of our model is that, in contrast to the conventional RCPSP model, activity durations are considered as variables, which makes it possible to deal with some realistic situations such as activity-crashing; i.e., the duration of an activity is not strictly fixed, but can be shortened by allocating an extra cost. This time/cost trade-off can be taken into account in our model via time-lag cost between the start time and the completion time of an activity.

Among the large literature on the RCPSP, a limited number of papers deal with convex objective functions. oucke et al (Vanhoucke et al (2001)) have proposed a branch-and-bound algorithm for the RCPSP to minimize the weighted earliness-tardiness. However, temporal constraints allowed in their problem are restricted to precedence constraints. For the RCPSP subject to general temporal constraints, Schwindt has proposed a branch-and-bound algorithm in Schwindt (1999). Also, he and his co-authors have devised local search algorithms in Neumann et al (2002) and Neumann et al (2003), where their target is not restricted to a convex objective function, but is more general.

It is known that (i) if there is no resource constraint, the RCPSP/conv can be formulated as the dual problem of a minimum cost flow problem, and hence can be solved efficiently, and (ii) resource conflicts can be removed by introducing precedence constraints between those activities that require common scarce resources. In this paper, we propose a metaheuristic algorithm exploiting these two properties. In our algorithm, we try to find a collection of additional precedence constraints that replace the resource constraints when the total time-lag cost is minimized. To search sets of additional precedence constraints efficiently, we employ an encoded solution-representation. We also incorporate an operation that often improves the schedule quality by adjusting the start/completion times of activities without violating resource and temporal constraints. This operation can be implemented by using a maximum flow algorithm.

To evaluate the performance of our algorithm, we conduct computational experiments. The computational results for the following three types of instances are reported: (i) benchmark instances of the RCPSP to minimize earliness-tardiness costs, (ii) instances in which activity-crashing is allowed, and (iii) instances in which temporal constraints can be violated with an extra cost.

9.2 Problem formulation

In this section, we give definitions and notations to formulate our problem, which we call the RCPSP to minimize convex time-lag costs, abbreviated to the RCPSP/conv.

Let R be a set of resources, and I be a set of activities. Each resource $r \in R$ is renewable in the sense that a prespecified amount K_r (≥ 0) is available in each period regardless of what amount has been used before. (Throughout this paper, we assume that the time horizon is discretized as in the standard formulation of the RCPSP.) Once activity i is initiated, it continues for d_i time periods without interruption (no preemption is allowed). One of the features of the RCPSP/conv is that the *duration* (or *processing time*) d_i of an activity i is not a constant value, but a variable in the range $[d_i^{\min}, d_i^{\max}]$, where $d_i^{\min}$ and $d_i^{\max}$ are input data with $0 < d_i^{\min} \leq d_i^{\max}$. An activity i requires a constant amount k_{ir} of resource r while being processed, where the value of k_{ir} is independent of the duration d_i. Therefore, the total amount of resources required by activity i is proportional to d_i.

For each activity $i \in I$, let i^s and i^c denote the events representing the start and the completion of activity i, respectively. We introduce a fictitious event 0 that represents the beginning of the project, and define the set of all events by $E = E^s \cup E^c \cup \{0\}$, where $E^s = \{i^s \mid i \in I\}$ and $E^c = \{i^c \mid i \in I\}$. Let x_v denote the time at which event v occurs ($v \in E$), and we assume that $x_0 = 0$; i.e., the project begins at time zero. For convenience, we sometimes denote the start time x_{i^s} (resp., the completion time x_{i^c}) of activity i by x_i^s (resp., x_i^c). A solution to the RCPSP/conv, or a *schedule*, is represented by a vector $\boldsymbol{x} = (x_v \mid v \in E)$. As the time horizon is discretized, we assume that the minimum and the maximum durations $d_i^{\min}$ and $d_i^{\max}$, as well as the start and the completion times x_i^s and x_i^c, of each activity $i \in I$, are all integers.

A schedule $\boldsymbol{x}$ is called *feasible* if it satisfies the following constraints.

Resource constraints

For each resource $r \in R$ and each time period $[t, t+1)$, the total amount of the resource r required by those activities $i \in I$ being processed in the period $[t, t+1)$ must not exceed the available amount K_r:

$$\sum_{i: x_i^s \leq t < x_i^c} k_{ir} \leq K_r.$$

Temporal constraints

A temporal constraint between events u and v is given by an inequality:

$$x_v - x_u \geq \delta_{uv} \tag{9.1}$$

on the event times x_u and x_v, where δ_{uv} is a (possibly negative) constant integer. (The temporal constraint is often referred to as *generalized precedence constraint*, since (9.1) can represent a precedence relation between activities i and j; i.e., $x_i^c \leq x_j^s$.) If δ_{uv} is negative, then $-\delta_{uv}$ gives the *maximum time-lag* between the two event times x_u and x_v; i.e., event u must be within $-\delta_{uv}$ time periods after event v occurs. Since event 0, which represents the beginning of the project, precedes all other events, it holds that $\delta_{0v} = 0$ for any $v \in E$.

We suppose that, in the RCPSP/conv, the minimum and maximum activity durations are specified via temporal constraints. More precisely, as the duration d_i of activity i is given by $d_i = x_i^c - x_i^s$, we describe the constraint that $d_i^{\min} \leq d_i \leq d_i^{\max}$ by two temporal constraints with $\delta_{i^s i^c} = d_i^{\min}$ and $\delta_{i^c i^s} = -d_i^{\max}$, respectively.

In the following, we consider $\delta_{uv} = -\infty$ if no temporal constraint with δ_{uv} is specified. Let $T \subseteq E \times E$ denote a set of event pairs (u, v) with $\delta_{uv} > -\infty$.

Another feature of the RCPSP/conv is its objective function. We introduce *time-lag costs*, which are charged according to the difference between two event times. In the RCPSP/conv, a set $C \subseteq E \times E$ of event pairs is specified together with a convex function $f_{uv}(\delta)$ for each $(u, v) \in C$. Given a feasible schedule $\boldsymbol{x}$, for each $(u, v) \in C$, we set $\delta = x_v - x_u$ and charge the time-lag cost $f_{uv}(\delta)$, where the value of $f_{uv}(\delta)$ makes sense only in the range of $\delta \in [\delta_{uv}, -\delta_{vu}]$, since $\delta_{uv} \leq x_v - x_u \leq -\delta_{vu}$ holds for any feasible $\boldsymbol{x}$. Then, the objective of the RCPSP/conv is to minimize the total time-lag costs:

$$f(\boldsymbol{x}) = \sum_{(u,v) \in C} f_{uv}(x_v - x_u). \tag{9.2}$$

Since event times x_v ($v \in E$) are all integers, we suppose that every time-lag cost function f_{uv} is given as a piece-wise linear convex function.

To finish this section, we present three examples that sometimes appear in real situations and can be dealt with in the RCPSP/conv.

Earliness-tardiness

Given the due date $\bar{d}_i$ of an activity i, its earliness-tardiness is defined by $|c_i - \bar{d}_i|$. By using event 0, the earliness-tardiness can be represented as a time-lag cost between the time of event 0 and the completion time of i. (Recall that event 0 represents the beginning of the project.) The time-lag cost function is given by

$$f_{0i^c}(\delta) = |\delta - \bar{d}_i|.$$

Activity-crashing

Activity-crashing is a technique to reduce the project duration (also called makespan) by shortening the durations of critical activities by allocating an extra cost. Since activity durations are not fixed in the RCPSP/conv, this time/cost trade-off can naturally be described with a time-lag cost between the start i^s and the completion i^c of activity i. Let $d^{\max}$ and $d^{\min}$ denote the normal and crash duration of activity i, respectively. Suppose that cost c_i is required to shorten the duration of i by one time period. Then, the time-lag cost function for activity-crashing is given by

$$f_{i^s i^c}(\delta) = c_i(d^{\max} - \delta).$$

Fast-tracking

Another way to reduce the project duration is to process in parallel two or more activities that are supposed to be done sequentially; i.e., there is a precedence relation between the activities. This technique corresponds to relaxing the precedence constraint and penalizing for the parallel processing. If we can consider that the penalty increases convexly as the amount of their overlaps increases, it is described in the RCPSP/conv.

Example. Consider a project involving one resource r and five activities $i = 1, 2, \ldots, 5$, as an example. (This example will be used later to illustrate our algorithm.) The available amount K_r of the resource r is two. The minimum and maximum durations, $d_i^{\min}$ and $d_i^{\max}$, and the resource requirements k_{ir} of the activities i are given as follows:

$$d_i^{\min} = 1, \qquad i = 1, 2, \ldots, 5,$$

$$d_i^{\max} = \begin{cases} 1, & i = 1, 4, 5, \\ 2, & i = 2, 3, \end{cases}$$

$$k_{ir} = \begin{cases} 1, & i = 1, 2, 3, 4, \\ 2, & i = 5. \end{cases}$$

Suppose that the following temporal constraints are imposed. Activity 1 must complete no later than activities 2 and 3 complete ($\delta_{1^c 2^c} = \delta_{1^c 3^c} = 0$).

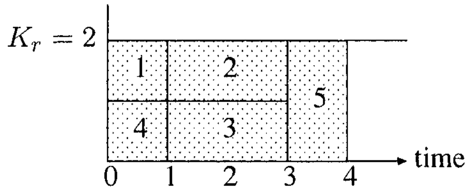


Figure 9.1. An optimal schedule of the example

Activity 4 must complete before activity 3 starts ($\delta_{4^c3^s} = 0$). Activity 5 cannot start before all other activities complete ($\delta_{i^c5^s} = 0$, $i \neq 5$). Activity 5 must be initiated within two time periods after activity 4 completes ($\delta_{5^s4^c} = -2$).

As for the objective function, we consider the following time-lag costs:

$$f_{i^si^c}(\delta) = 2 - \delta, \quad i = 2, 3,$$
$$f_{4^c5^s}(\delta) = 2 - \delta,$$
$$f_{05^c}(\delta) = \max\{0, \delta - 4\}.$$

The first time-lag cost means that if we finish activity i = 2 or 3 in the crashed duration (i.e., $d_i^{\min} = 1$), a unit cost is charged. The second cost indicates that it is desirable to have a longer interval between activities 4 and 5. The last one is to complete activity 5 (hence all the activities) within four time periods.

As shown by the Gantt chart in Figure 9.1, there is a schedule with a zero time-lag cost.

9.3 Project network

To represent temporal relations between activities, an *activity-on-node project network* N is commonly used (e.g., Neumann et al (2002)). This is a directed network whose node set is given by the activity set I, and whose arc set represents the temporal relations on the activities. Since, in the RCPSP/conv, the start times x_i^s and the completion times x_i^c of activities i are both decision variables and temporal constraints are defined on event basis, we introduce an *event-on-node project network* N defined by (E, T, δ) so that the event set E is associated with the node set, and each temporal constraint $(u, v) \in T$ corresponds to an arc (u, v) of length δ_{uv}.

In the subsequent arguments, we assume that project network N contains no positive-length cycle, since otherwise, no schedule x can satisfy all temporal constraints simultaneously. Then, we can calculate the longest path length $dist(u, v)$ from any node u to any node v. (If v is unreachable from u, $dist(u, v)$ is defined as $-\infty$.) Any feasible schedule x must satisfy $x_v - x_u \geq dist(u, v)$ for any $u, v \in E$. It is known that if N has no positive-length cycle, there exists

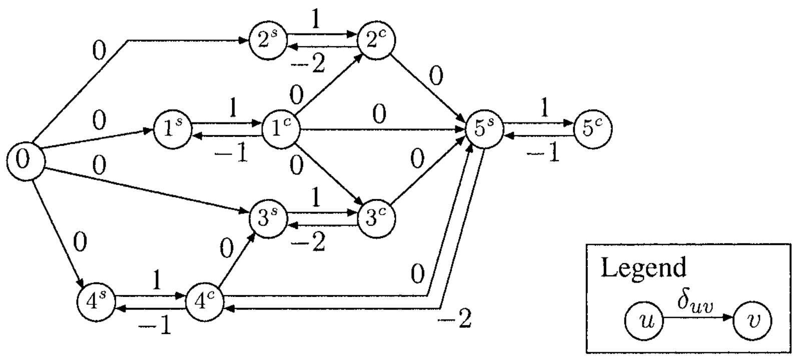


Figure 9.2. The event-on-node project network of the example

a schedule that satisfies all the temporal constraints (resource constraints are ignored in this argument); e.g., take a schedule x with $x_v = dist(0, v)$, $v \in E$.

We define a binary relation $\prec$ on the event set E by

$$u \prec v \Leftrightarrow dist(u, v) \geq 0 \text{ and } dist(v, u) < 0. \qquad (9.3)$$

(This relation is called *distance order* in Neumann et al (2002), although it is defined on the activity set there, rather than the event set, since the activity-on-node project network is used.) If $u \prec v$ holds, it means that event v cannot be prior to event u, but u can be prior to v. Relation $\prec$ is irreflexive (i.e., $\forall v \in E$, $v \not\prec v$) and transitive (i.e., $\forall u, v, w \in E$, $u \prec v$ and $v \prec w \Rightarrow u \prec w$).

All values of $dist(u, v)$ can be calculated, or a positive-length cycle is detected, in polynomial time, e.g., in $O(n^3)$ time by the Floyd-Warshall algorithm, and in $O(n(m + n \log n))$ time by a repeated shortest path algorithm (Ahuja et al (1993)), where $n = |E|$ is the number of nodes and $m = |T|$ is the number of arcs in the network N. While the Floyd-Warshall algorithm is easy to implement, it is not suited for sparse networks. Since N is usually sparse for practical RCPSP/conv instances, the latter algorithm is preferable for our purpose.

Example (cont'd). Figure 9.2 depicts the event-on-node project network N of our example, where some of the arcs from event 0 are omitted.

9.4 Algorithm

9.4.1 Basic idea

To design an algorithm for the RCPSP/conv, we first consider the problem PSP/conv, which seeks to minimize the total convex time-lag cost $f(x)$ subject to only temporal constraints (i.e., no resource constraint is imposed). It is known that PSP/conv is reduced to the dual problem of a minimum cost flow problem, and hence its optimal schedule x^* can be computed efficiently; see, e.g., Ahuja et al (1993), Ahuja et al (1999). (Although we do not explain the details here, we emphasize that each temporal constraint $(u, v) \in T$ of PSP/conv is associated with an arc (u, v) in the network of the corresponding minimum cost flow problem, and the flow on arc (u, v) is interpreted as the dual variable of temporal constraint (u, v) in PSP/conv.) Of course, x^* may contain resource conflicts among some activities including i and j, but such conflicts can be resolved by introducing precedence constraints such as $x_i^c \leq x_j^s$. Even after introducing such additional precedence constraints, PSP/conv is still tractable. The main idea of our algorithm is, instead of using resource constraints, to introduce a collection of precedence constraints defined appropriately. (We should note here that this idea has already been adopted in several existing algorithms, particularly, in branch-and-bound algorithms, e.g., Bartusch et al (1988), De Reyck and Herroelen (1998).)

Let $P \subseteq E^c \times E^s$ be a set of precedence constraints to be added for the purpose of replacing resource constraints, where each element $(i^c, j^s) \in P$ is identified with the precedence constraint $x_i^c \leq x_j^s$. As mentioned above, the RCPSP/conv with P but without resource constraints is formulated as a problem of PSP/conv, and we denote this problem as PSP/conv(P).

We now discuss a condition for P to replace the resource constraints. A set F of activities such that

$$\exists r \in R, \quad \sum_{i \in F} k_{ir} > K_r$$

is called a *forbidden set* (i.e., the resource constraints forbid processing in parallel all the activities in F). If P satisfies the following condition, then any feasible solution x to PSP/conv(P) satisfies the resource constraints, and thus x is feasible for the original RCPSP/conv:

For any forbidden set $F \subseteq I$, P contains at least one precedence constraint (i^c, j^s) related to some $i, j \in F$. (9.4)

In this case, we say that P is *resource-feasible*. On the other hand, for any feasible solution x to the original RCPSP/conv, there exists a resource-feasible set of precedence constraints P such that x is feasible for PSP/conv(P); for instance, $P = \{(i^c, j^s) \in E^c \times E^s \mid x_i^c \leq x_j^s\}$ satisfies this property. Therefore, our problem becomes to find a resource-feasible set of precedence constraints P^* such that PSP/conv(P^*) minimizes the objective function. We try to find such P^* by a local search based heuristic algorithm.

9.4.2 Search space

Local search has widely been applied to solve hard combinatorial optimization problems. It starts with an initial solution and repeatedly modifies it so that the solution quality is improved. To embody local search, we define below the search space, the evaluation function and the neighborhood.

In our algorithm, we do not represent a solution explicitly by using a set of precedence constraints P, partly because $O(|I|^2)$ time and space are required to store and manipulate a solution in this straightforward approach, and partly because it is not easy to achieve an efficient checking of condition (9.4) (there are too many forbidden sets in general).

To overcome these difficulties, we adopt an encoded representation using an *event-list* $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{|E|-1})$, which is a permutation of all events v in $E \setminus \{0\}$ such that $i^s <_\sigma i^c$ for any $i \in I$, where $u <_\sigma v$ $(u, v \in E \setminus \{0\})$ means that event u appears prior to v in event-list σ. Then, given an event-list σ, we decode it to a set of precedence constraints P_σ as follows:

$$P_\sigma = \{(i^c, j^s) \in E^c \times E^s \mid i^c <_\sigma j^s\}.$$

If P_σ is resource-feasible, we also say that the event-list σ is resource-feasible, because any schedule obeying P_σ is guaranteed to satisfy the resource constraints. (In Section 9.4.3, we will explain how to check the resource-feasibility of P_σ.) Notice that PSP/conv(P_σ) may be infeasible; i.e., no solution satisfies all the original temporal constraints T and the additional precedence constraints P_σ. This corresponds to the case that adding all arcs $(i^c, j^s) \in P_\sigma$ of length 0 to the original project network N creates a positive-length cycle. If an event-list σ is resource-feasible and furthermore PSP/conv(P_σ) is feasible, then we say that σ is *feasible*. Note that the problem of determining whether there exists a feasible event-list σ is NP-hard, because it is equivalent to determining the feasibility of the given RCPSP/conv instance.

This fact indicates that it should be inappropriate to define the search space of the local search algorithm as consisting of only feasible event-lists. Therefore, instead, we develop a local search algorithm operating with resource-feasible event-lists σ that satisfy the following condition, where $\prec$ was defined in (9.3):

$$i^c \prec j^s \Rightarrow i^c <_\sigma j^s, \qquad i, j \in I. \tag{9.5}$$

This condition reduces the size of the search space, while retaining the existence of an optimal event-list σ^* in the search space. In the following, we call a resource-feasible event-list σ *valid* if it satisfies condition (9.5), and we sometimes refer to a valid event-list as a solution of our local search algorithm.

If a solution σ is feasible, it is evaluated by the total time-lag cost (9.2) of PSP/conv(P_σ) (a smaller cost is better). To evaluate feasible and infeasible solutions in a unified way, we consider the (originally given) temporal constraints T of PSP/conv(P_σ) as soft constraints, so that it always has a feasible solution. More precisely, for $(u, v) \in T$, we introduce a time-lag cost with $f_{uv}(\delta) = M \cdot \max\{\delta_{uv} - \delta, 0\}$, where M is a sufficiently large value, and then set the δ_{uv} of the temporal constraint to $-\infty$. Then, if an event-list σ

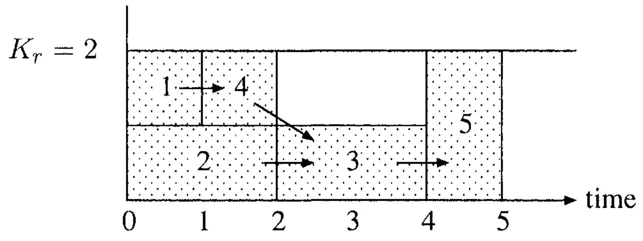


Figure 9.3. An optimal schedule of PSP/conv(P_σ), where $\sigma = (2^s, 1^s, 1^c, 4^s, 2^c, 4^c, 3^s, 3^c, 5^s, 5^c)$

is feasible (resp., infeasible), the minimum cost of PSP/conv(P_σ) is less than (resp., greater than or equal to) M. As an exceptional rule, however, we consider the temporal constraints on activity durations (i.e., $d_i^{\min} \leq d_i \leq d_i^{\max}$) as hard constraints as in the original RCPSP/conv. (This rule does not make PSP/conv(P_σ) infeasible.)

Let σ be a resource-feasible event-list with $i^c <_\sigma j^s$; i.e., $(i^c, j^s) \in P_\sigma$. If there is some activity k such that $i^c <_\sigma k^s <_\sigma k^c <_\sigma j^s$, then precedence constraint (i^c, j^s) is redundant in the sense that the resource-feasibility of P_σ is ensured without (i^c, j^s). In evaluating a solution σ, we take into account only non-redundant precedence constraints in P_σ to reduce the computational effort of solving PSP/conv(P_σ).

Example (cont'd). Consider an event-list $\sigma = (2^s, 1^s, 1^c, 4^s, 2^c, 4^c, 3^s, 3^c, 5^s, 5^c)$. Then, P_σ consists of precedence constraints $(1^c, 4^s)$, $(1^c, 3^s)$, $(1^c, 5^s)$, $(2^c, 3^s)$, $(2^c, 5^s)$, $(4^c, 3^s)$, $(4^c, 5^s)$ and $(3^c, 5^s)$. Since P_σ satisfies conditions (9.4) and (9.5), this event-list σ is valid. An optimal solution to PSP/conv(P_σ) is illustrated by the Gantt chart in Figure 9.3, where non-redundant precedence constraints in P_σ are shown by directed arcs. The total time-lag cost of this schedule is $f_{2^s 2^c}(2-0) + f_{3^s 3^c}(4-2) + f_{4^c 5^s}(4-2) + f_{0 5^c}(5-0) = 0+0+0+1 = 1$.

9.4.3 Neighborhood

To define the neighborhood $\mathcal{N}(\sigma)$ of a solution σ, we introduce an operation MOVEBACKWARD(i, j). This operation is defined for a pair (i, j) of activities such that $i^c <_\sigma j^s$ and $i^c \not\prec j^s$, and modifies event-list σ by moving event j^s to immediately before i^s so that precedence constraint (i^c, j^s) is eliminated from P_σ. To ensure that the resultant event-list satisfies condition (9.5), events v ($= k^s$ or k^c, $k \in I$) such that $i^s <_\sigma v <_\sigma j^s$ are also moved to the position immediately before i^s if $k^c \prec j^s$ holds. The relative positions among the moved events are preserved.

Example (cont'd). By applying operation MOVEBACKWARD$(2,3)$ to $\sigma = (2^s, 1^s, 1^c, 4^s, 2^c, 4^c, 3^s, 3^c, 5^s, 5^c)$ used in the above example, we obtain an event-list $(4^s, 4^c, 3^s, 2^s, 1^s, 1^c, 2^c, 3^c, 5^s, 5^c)$. Notice that two events 4^s and 4^c were moved to before 2^s together with 3^s, because $4^c \prec 3^s$ holds.

Let σ' denote the list obtained by applying MOVEBACKWARD(i,j) to σ. Since σ' may not be resource-feasible, we subsequently apply procedure MAKEVALID described below, which checks the resource-feasibility of σ' and repairs it if necessary. While scanning events in list σ' from the beginning, MAKEVALID constructs the set A of activities that are allowed to be processed in parallel under $P_{\sigma'}$; starting with $A = \emptyset$, we add activity i into A (resp., remove i from A) if the next event is i^s (resp., i^c). Whenever A becomes a forbidden set (i.e., condition (9.4) is violated), we try to introduce a precedence constraint (i_1, i_2) for some $i_1, i_2 \in A$ by changing the position of i_1^c in σ' to immediately before i_2^s. We select as i_2^s the start event that appears last among those in A. As to i_1^c $(i_1 \neq i_2)$, we consider one that satisfies $dist(i_2^s, i_1^c) \leq 0$, since otherwise, σ' becomes infeasible as a result of adding precedence constraint (i_1, i_2). If there is more than one such i_1^c, we take the one that appears first in σ'; otherwise (if there is no such i_1^c), we change the position of i_2^s so that it appears before any other start event i^s of $i \in A$, by applying MOVEBACKWARD(j, i_2), where j is the activity whose start event j^s appears first among those activities in A.

Since procedure MAKEVALID is not guaranteed to terminate in a finite number of steps because of the last modification by MOVEBACKWARD, we set a limit to the number of applications of MOVEBACKWARD. Once this limit is exceeded, MAKEVALID ignores in the rest of the computation the condition $dist(i_2^s, i_1^c) \leq 0$ if no i_1 satisfies it. In this case, MAKEVALID outputs an infeasible event-list σ'', but σ'' is guaranteed to be valid. (We set this limit to ten, but it had rarely been exceeded in our experiments.) Procedure MAKEVALID is summarized below.

MAKEVALID
Input: an event-list σ' on $E \setminus \{0\}$
Output: a valid event-list σ''

Step 1 (Initialization)
Set $A := \emptyset$ and $k := 1$. Event-list σ'' is initially empty.

Step 2 (Next event)
If $k > |E|$, then return σ'' and terminate. Otherwise, let the next event $v := \sigma'(k)$. If v is a start event, say i^s, go to Step 3. If v is a completion event, say i^c, go to Step 5.

Step 3 (Insertion of start event)
Update $A := A \cup \{i\}$. If A is not a forbidden set, then set $\sigma''(k) := i^s$, $k := k + 1$ and return to Step 2; otherwise, go to Step 4.

Step 4 (Modification of event-list σ')
Let $i_2 := i$, and let A' be the set of activities $j \in A \setminus \{i_2\}$ such that $dist(i_2^s, j^c) \leq 0$ holds.

Case 1. If A' is not empty, let $i_1 \in A'$ be the one whose completion event i_1^c appears first in σ'. (Note that $i_2^s <_{\sigma'} i_1^c$ always holds here.) Modify σ' by moving i_1^c to immediately before i_2^s (as a result of this, the k-th event $\sigma'(k)$ is changed from i_2^s to i_1^c). Remove i_2 from A, and return to Step 2.

Case 2. If A' is empty, check if MOVEBACKWARD has been applied the predetermined number of times so far. If yes, let $A' := A \setminus \{i_2\}$ and apply Case 1; otherwise, let $j \in A \setminus \{i_2\}$ be the one whose start event j^s appears first in σ'. Modify σ' by applying MOVEBACKWARD(j, i_2). Return to Step 1.

Step 5 (Insertion of completion event)
Let $\sigma''(k) := i^c$, and update $A := A \setminus \{i\}$. Let $k := k + 1$ and return to Step 2.

Example (cont'd). Let us consider the process of MAKEVALID applied to $\sigma' = (4^s, 4^c, 3^s, 2^s, 1^s, 1^c, 2^c, 3^c, 5^s, 5^c)$, which was obtained from the event list σ by MOVEBACKWARD$(2, 3)$. Starting with $A := \emptyset$ and $k := 1$, we scan the first event 4^s and add activity 4 to the set A. Since the next event is 4^c, activity 4 is immediately removed from A, and A becomes empty again. Then, by repeating Steps 2 and 3, we get a forbidden set $A = \{3, 2, 1\}$ at Step 3 when $k = 5$ holds, and go to Step 4. Here, $i_2 = 1$ and $A' = \emptyset$, because $dist(1^s, 3^c) = dist(1^s, 2^c) = 1$. Then, Case 2 of Step 4 applies, and σ' is modified by MOVEBACKWARD$(3, 1)$; σ' becomes $(4^s, 4^c, 1^s, 3^s, 2^s, 1^c, 2^c, 3^c, 5^s, 5^c)$. We restart the process from Step 1 with the new σ', and we have a forbidden set $A = \{1, 3, 2\}$ again when $k = 5$ holds. At this time, $i_2 = 2$ and $A' = \{1\}$, and Case 1 applies. Then, σ' becomes $(4^s, 4^c, 1^s, 3^s, 1^c, 2^s, 2^c, 3^c, 5^s, 5^c)$. We remove activity 2 from A and continue to scan the list σ' from $k = 5$ with $A = \{1, 3\}$. In the rest of the computation, Step 4 is not executed, and A changes as follows: $\{1, 3\} \rightarrow \{3\} \rightarrow \{3, 2\} \rightarrow \{3\} \rightarrow \emptyset \rightarrow \{5\} \rightarrow \emptyset$. Finally, we obtain a valid event-list $\sigma'' = (4^s, 4^c, 1^s, 3^s, 1^c, 2^s, 2^c, 3^c, 5^s, 5^c)$. An optimal schedule of PSP/conv$(P_{\sigma''})$ is shown in Figure 9.4. The total time-lag cost of this schedule is $f_{2^s 2^c}(3-2) + f_{3^s 3^c}(3-1) + f_{4^c 5^s}(3-1) + f_{0 5^c}(4-0) = 1 + 0 + 0 + 0 = 1$.

To improve the quality of σ'' further, we consider the following operation, which tries to reduce the number of precedence constraints in $P_{\sigma''}$ (i.e., en-

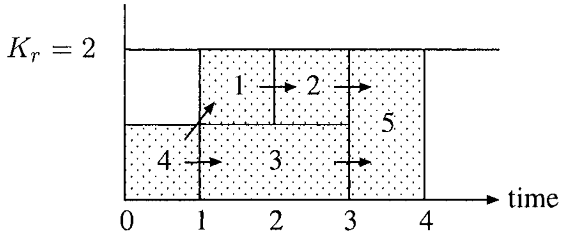


Figure 9.4. An optimal schedule of PSP/conv(P''_σ), where $\sigma'' = (4^s, 4^c, 1^s, 3^s, 1^c, 2^s, 2^c, 3^c, 5^s, 5^c)$

large the feasible region of the problem PSP/conv($P_{\sigma''}$)) without violating the resource-feasibility of σ''. Suppose that there is a pair of events i^c and j^s in σ'' such that (i) j^s is the first start event such that $i^c <_{\sigma''} j^s$, and (ii) $i^c \not\prec j^s$ and also $\tilde{i}^c \not\prec j^s$ for any $\tilde{i}^c$ ($i^c <_{\sigma''} \tilde{i}^c <_{\sigma''} j^s$). Then, we move j^s to just before i^c if it does not break the resource-feasibility of σ''. By this operation, precedence constraint (i^c, j^s) is removed from $P_{\sigma''}$. Although we omit the detail, this operation can be incorporated within MAKEVALID.

Example (cont'd). A valid event-list $\sigma'' = (4^s, 4^c, 1^s, 3^s, 1^c, 2^s, 2^c, 3^c, 5^s, 5^c)$ is improved by moving 1^s to immediately before 4^c. The resultant event-list $(4^s, 1^s, 4^c, 3^s, 1^c, 2^s, 2^c, 3^c, 5^s, 5^c)$ is still valid, and brings the optimal schedule shown in Figure 9.1.

In a symmetric way, we define an operation MOVEFORWARD(i, j), which moves event i^c to immediately after j^c together with all events $v = k^s$ or k^c such that $i^c <_\sigma v <_\sigma j^c$ and $i^c \prec k^s$. In order to make the resultant event-list valid, we apply the reverse version of MAKEVALID, which works symmetrically to MAKEVALID, scanning the event-list backward from the end.

The neighborhood $\mathcal{N}(\sigma)$ of solution σ is now defined as the set of event-lists σ'' that can be obtained from σ by applying MOVEBACKWARD followed by MAKEVALID, or by applying MOVEFORWARD followed by the reverse version of MAKEVALID. Starting with an initial solution $\sigma = \sigma^0$ (how to generate σ^0 will be explained in the next subsection), our local search repeats replacing the current solution σ with a better one σ'' in the neighborhood $\mathcal{N}(\sigma)$, until there exists no better solution in $\mathcal{N}(\sigma)$. We adopt the so-called first-move strategy; the current solution σ is replaced with the better neighbor $\sigma'' \in \mathcal{N}(\sigma)$ found first.

9.4.3.1 Neighborhood reduction.

To evaluate each neighbor $\sigma'' \in \mathcal{N}(\sigma)$ of the current solution σ, we solve PSP/conv($P_{\sigma''}$) by applying a minimum cost flow algorithm. The networks of these neighbors have similar structures, and therefore, we do not solve the problems from scratch each time, but calculate optimal values by using a re-optimization technique based on a primal-dual algorithm for the minimum cost flow problem. Since this computation is still rather expensive, however, it is not efficient to search the whole neighborhood in every iteration of the local search. To make the search more effective, we reduce the neighborhood size as follows.

As mentioned above, $\textsc{MoveBackward}(i,j)$ and $\textsc{MoveForward}(i,j)$ are designed to remove precedence constraint (i^c, j^s) from P_σ. Let x be an optimal solution to PSP/conv(P_σ). Duality of linear programming tells us that, in order to decrease the total time-lag cost $f(x)$, it is necessary to remove some precedence constraint $(i^c, j^s) \in P_\sigma$ such that the corresponding optimal dual variable is positive (i.e., the optimal flow corresponding to x is positive on arc (i^c, j^s)). In our local search, we apply $\textsc{MoveBackward}(i,j)$ and $\textsc{MoveForward}(i,j)$ only for such precedence constraints. The neighborhood reduced in this way is denoted by $\tilde{\mathcal{N}}(\sigma)$.

9.4.3.2 Speed up in finding a feasible solution. Our local search algorithm sometimes requires many replacements of the solution before reaching the first feasible one. (Recall that it is NP-hard to find a feasible schedule.) To reduce computational time spent in this phase, before reaching a feasible schedule, we do not take into account the time-lag cost in evaluating the quality of a solution σ; i.e., we consider only the amount of infeasibility of the schedule. Once we obtain a feasible solution, we then evaluate solutions exactly by their total time-lag costs. This simple strategy makes the local search algorithm run faster in the early stage.

9.4.4 Initial solution

To prepare an initial solution for the local search, we first ignore all the resource constraints and solve the resultant PSP/conv($\emptyset$). (If this problem is infeasible, then we can conclude that the original RCPSP/conv instance is infeasible.) Let x^0 be its optimal solution and σ^0 be an event-list obtained by sorting the events $v \in E \setminus \{0\}$ in non-decreasing order of their event times x^0_v. Ties are broken randomly subject to condition (9.5). Since σ^0 is not valid in general, we then apply $\textsc{MakeValid}$ to σ^0 and use the output valid event-list as an initial solution.

9.4.5 Further improvement by shift operations

The local search terminates when no better solution is found in the neighborhood $\tilde{\mathcal{N}}(\sigma)$ of the current solution σ. In this situation, we call σ and the optimal solution x to PSP/conv(P_σ) locally optimal. In our algorithm, we then incorporate another type of operation, called *shift operation*, hoping to improve a locally optimal solution x further. Contrast to MOVEBACKWARD, MOVEFORWARD and MAKEVALID, a shift operation modifies a schedule x directly rather than via modifying the encoded solution σ. We say that, for a schedule x, we left-shift (resp., right-shift) an event v if we decrease (resp., increase) its event time x_v. Let $E' \subset E$ be a set of events such that $0 \notin E'$, where event 0 represents the beginning of the project. Then, shift operation LEFTSHIFT(E') (resp., RIGHTSHIFT(E')) applied to x simultaneously left-shifts (right-shifts) all events $v \in E'$ by one time period; i.e., $x_v := x_v - 1$ (resp., $x_v := x_v + 1$), $v \in E'$. We explain only LEFTSHIFT in the following, but the symmetric argument applies to RIGHTSHIFT.

Given a feasible schedule x, while keeping its feasibility, we try to decrease its total time-lag cost $f(x)$ by applying LEFTSHIFT(E') for some E'. Let $x(E')$ denote the schedule obtained from x by LEFTSHIFT(E'). Schedule $x(E')$ is feasible if and only if E' satisfies the following conditions:

(C1) For each temporal constraint $(u, v) \in T$ such that $x_v - x_u = \delta_{uv}$, E' satisfies $v \in E' \Rightarrow u \in E'$.

(C2) For each time t, let E_t^s and E_t^c denote the set of start events and completion events v such that $x_v = t$, respectively. For any time period $[t-1, t)$ and resource $r \in R$,

$$\sum_{i^s \in E_t^s \cap E'} k_{ir} + \sum_{j^c \in E_t^c \setminus E'} k_{jr} + \sum_{i:\, x_i^s < t < x_i^c} k_{ir} \le K_r.$$

The problem of determining if there exists an E' such that $x(E')$ is a feasible schedule with $f(x(E')) < f(x)$ can be proved to be NP-hard. Therefore, we solve it heuristically. Suppose that the events in E_t^s and E_t^c, respectively, are numbered arbitrarily, say $E_t^s = \{i_{t,1}^s, i_{t,2}^s, \ldots, i_{t,L}^s\}$ and $E_t^c = \{j_{t,1}^c, j_{t,2}^c, \ldots, j_{t,M}^c\}$. Then, we left-shift some number (possibly zero) of the first events in E_t^s and E_t^c, say, $\{i_{t,1}^s, \ldots, i_{t,l}^s\}$ and $\{j_{t,1}^c, \ldots, j_{t,m}^c\}$. In other words, we restrict a set of events E' to those satisfying the following condition in addition to (C1) and (C2):

(C3) For any t,

$$i_{t,l+1}^s \in E' \Rightarrow i_{t,l}^s \in E', \quad l = 1, 2, \ldots, L-1,$$

and

$$j_{t,m+1}^c \in E' \Rightarrow j_{t,m}^c \in E', \quad m = 1, 2, \ldots, M-1.$$

Under this restriction, as described below, the problem of minimizing $f(x(E'))$ is solvable in polynomial time. If an optimal set E^* in this sense satisfies $f(x(E^*)) < f(x)$, we can improve x by applying LEFTSHIFT(E^*).

To minimize $f(\boldsymbol{x}(E'))$ subject to (C1)–(C3) for a given feasible schedule $\boldsymbol{x}$, we construct a directed graph G whose node set is the event set E and whose arcs, each having a weight, are given as follows.

- For each condition $u \in E' \Rightarrow v \in E'$ in (C1) and (C3), arc (v, u) with weight $w(v, u) = \infty$ is introduced.
- For each time period $[t-1, t)$ and $l \in \{1, 2, \ldots, L\}$, arc $(i^s_{t,l}, j^c_{t,m})$ is introduced for the maximum $m \in \{1, 2, \ldots, M\}$ (if any) such that, for some $r \in R$,

$$\sum_{i^s \in \{i^s_{t,1}, \ldots, i^s_{t,l}\}} k_{ir} + \sum_{j^c \in \{j^c_{t,m}, \ldots, j^c_{t,M}\}} k_{jr} + \sum_{i:\ x^s_i < t < x^c_i} k_{ir} > K_r$$

  (which means that some resource constraint will be violated if we left-shift the first l events in E^s_t and the first $m-1$ events in E^c_t). The weight of the arc is ∞.
- For each time-lag cost f_{uv} for $(u, v) \in C$, arc (u, v) with weight $w(u, v) = f_{uv}(x_v - x_u + 1) - f_{uv}(x_v - x_u)$ and arc (v, u) with weight $w(v, u) = f_{uv}(x_v - x_u - 1) - f_{uv}(x_v - x_u)$ are introduced.

Multiple arcs are replaced with a single arc with their total weight. Let $w(E')$ denote the sum of the weights of the arcs out-going from E', i.e., $w(E') = \sum_{u \in E', v \notin E'} w(u, v)$. A set E' of events satisfies conditions (C1)–(C3) if and only if $w(E') < \infty$ holds, and in this case, it holds that $w(E') = f(\boldsymbol{x}(E')) - f(\boldsymbol{x})$. Therefore, our objective is to find an optimal set that minimizes $w(E')$. Although graph G contains arcs with negative weights, the problem of minimizing $w(E')$ can be solved in the running time of finding a minimum cut in a directed graph without negative arc weights. This is because, by convexity of time-lag cost functions f_{uv}, it holds that $w(u, v) + w(v, u) \geq 0$ for any pair of u and v, and therefore, finding an optimal set E^* is reduced to finding a *most positive cut* in a directed graph. For more details, see, e.g., McCormick and Ervolina (1994).

Example (cont'd). Let $\boldsymbol{x}$ be the feasible schedule in Figure 9.4. We consider applying operation LEFTSHIFT to $\boldsymbol{x}$. To minimize $f(\boldsymbol{x}(E'))$ subject to conditions (C1)–(C3), we construct a directed graph G, which is depicted in Figure 9.5. Events in $E^s_1 = \{1^s, 3^s\}$ and $E^c_3 = \{2^c, 3^c\}$ are ordered according to the activity index. In this figure, thick solid, thick dashed and thick dotted arcs (all of which have a weight of infinity) correspond to conditions (C1), (C2) and (C3), respectively, and the other arcs represent the difference of the corresponding time-lag costs. An optimal set E^* with the minimum weight is $\{1^s, 1^c, 2^s\}$, as shown in Figure 9.5. Since $w(E^*) = -1$, we can improve

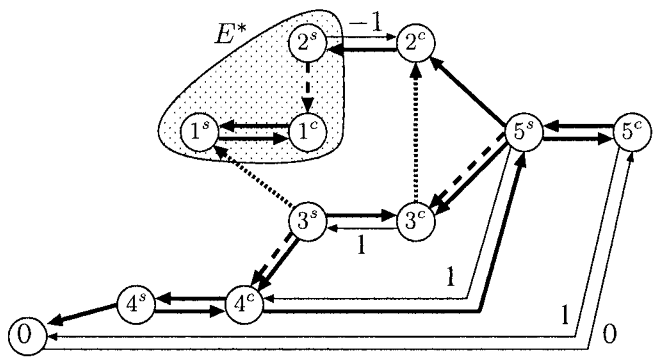


Figure 9.5. A directed graph G defined in LEFTSHIFT

the schedule x, and the total time-lag cost of the resultant schedule $x(E^*)$ is $f(x(E^*)) = f(x) + w(E^*) = 1 - 1 = 0$. ($x(E^*)$ is the optimal schedule shown in Figure 9.1.)

In our algorithm, when a locally optimal schedule x is obtained, and if it is feasible, we apply LEFTSHIFT and find an optimal set E^* that minimizes $f(x(E^*))$. We check if $f(x(E^*)) < f(x)$, and if yes, we replace x with $x(E^*)$. Then, we try RIGHTSHIFT next. We repeat this process as long as we can improve x by LEFTSHIFT or RIGHTSHIFT. If the locally optimal schedule x is improved at least once by a shift operation, we restart the local search from x, where, as in the preparation of an initial list σ^0 (Section 9.4.4), the encoded solution σ corresponding to x is generated by sorting event times and applying MAKEVALID. On the other hand, if the locally optimal schedule x cannot be improved by this try, our local search terminates.

9.4.6 Iterated Local Search

Since a single run of the local search is not powerful enough to find a good solution, we execute local search iteratively. Whenever the local search terminates (after trying shift operations), we go back to the best solution σ^* found so far, and perturb it to obtain a different solution σ, from which we start another run of local search. As an operation for the perturbation, we use MOVEBACKWARD(i_1, i_2) or MOVEFORWARD(i_1, i_2), followed by MAKEVALID (how to select a pair of activities (i_1, i_2) will be explained below). At the first neighborhood search of the subsequent run of local search, we do not apply operations MOVEBACKWARD(i_2, i_1) and MOVEFORWARD(i_2, i_1) with the pair of i_2 and i_1 to generate a neighbor, in order to avoid returning to the best solution σ^*.

Now we describe how to perturb σ^*. Among the candidates of (i_1, i_2), we try smaller perturbations first; we give higher priority to pairs (i_1, i_2) such that the difference of the positions of i_1^c and i_2^s in σ^* is smaller. To prevent applying the same operation to the same solution σ^*, in the iterated local search, we store in the memory the operations tested on σ^*, and reset the memory whenever a better solution than σ^* is found.

The entire process of our algorithm is summarized below.

ITERATEDLOCALSEARCH
Input: an RCPSP/conv instance
Output: a schedule x

Step 1 (Initialization)

Check if the project network N has a positive length cycle (Section 9.3). If yes, conclude that the instance is infeasible and terminates; otherwise, prepare an initial event-list σ (Section 9.4.4).

During the following process, whenever a feasible schedule with zero total time-lag cost is found, or a predetermined computational time has elapsed, output the best schedule x^* found by then and terminate.

Step 2 (Neighborhood search)

Find a better solution σ'' in the reduced neighborhood $\tilde{\mathcal{N}}(\sigma)$ (Sections 9.4.3). If there exists no such σ'', go to Step 3. Otherwise, update the current solution $\sigma := \sigma''$ and return to Step 2.

Step 3 (Shift operation)

(σ is now locally optimal.) Let x be an optimal solution to PSP/conv(P_σ). Try to improve x by shift operations (Section 9.4.5). If x is improved, then generate a solution σ corresponding to x and return to Step 2; otherwise, go to Step 4.

Step 4 (Perturbation)

Let σ^* be the best solution found so far. Perturb σ^* and obtain an initial solution σ for the next run of local search (Section 9.4.6). Return to Step 2.

9.5 Computational experiments

To evaluate the performance of our algorithm, we conducted computational experiments. Our algorithm uses a minimum cost flow algorithm and a maximum flow (or minimum cut) algorithm as subroutines for evaluating solutions and for the shift operation, respectively. For these algorithms, we use RelaxIV and HIPR, respectively. RelaxIV is a C++ solver based on a primal-dual algorithm called RELAX (Bertsekas and Tseng (1994)) and available from the

Table 9.1. Computational results for the RCPSP to minimize the weighted earliness-tardiness cost

Instance set	$\|I\| = 10$	$\|I\| = 20$	$\|I\| = 50$	$\|I\| = 100$
#feasible	73/73	70/70	73/73	70/78
Comparison with BB†	0-73-0	2-68-0	33-33-7	52-11-15
Comparison with LS†	—	—	37-32-4	41-12-25

† n_b-n_e-n_w means that our result is better than, equivalent to and worse than the result of BB/LS for n_b, n_e and n_w instances, respectively.

The time limit was 10 seconds on a 3.0GHz processor for our algorithm, and 100 seconds on a 333MHz processor for BB and LS.

CRIFOR web site at `http://sorsa.unica.it/`. HIPR is an efficient implementation of a push-relabel algorithm (Cherkassky and Goldberg (1997)) and copyrighted by IG Systems, Inc. (`http://www.igsystems.com/`). Our code is written in C++, and all experiments were conducted on a Dell Precision 470 with dual 3.0GHz Intel Xeon processors, where the code was run on a single processor.

9.5.1 Minimization of the weighted earliness-tardiness cost

To compare our algorithm with other existing ones, we solved the instances used in Neumann et al (2002). The objective of these instances is to minimize the weighted earliness-tardiness cost subject to general temporal constraints (rather than precedence constraints), where activity durations are fixed. These instances are based on benchmark instances generated by a problem generator named ProGen/max proposed in Schwindt (1998). (Benchmark instances with up to 1000 activities generated by ProGen/max are available through the Internet.) Since these benchmark instances contain neither the earliness and tardiness cost nor the due date, such additional data have been generated randomly for each instance. For our experiment, we take four instance sets, each containing 90 instances with 10, 20, 50 and 100 activities, respectively. The number of resources is five for all instances in all sets. Excluding those instances that have been proved to be infeasible (i.e., no schedule satisfies the resource constraints and the temporal constraints), we solved 73, 70, 73 and 78 instances, respectively, out of 90. We set the maximum computational time to 10 seconds for each instance.

The results are summarized in Table 9.1. The second row (#feasible) of this table shows, for each instance set, the number of instances for which our algorithm found a feasible schedule (out of the total number of feasible instances). The next two rows show the comparison results of our algorithm with the (trun-

Table 9.2. Effectiveness of the shift operation

Instance set	$\lvert I\rvert = 50$	$\lvert I\rvert = 100$
Comparison of ITERATEDLOCALSEARCH with the one not using shift operations	22-43-8	43-21-14

cated) branch-and-bound algorithm (denoted by BB) proposed in Schwindt (1999) and the local search algorithm (denoted by LS) whose idea is given in Neumann et al (2002) and Neumann et al (2003), respectively. Each entry in the format of n_b-n_e-n_w means that our result is better than, equivalent to and worse than the results of BB or LS for n_b, n_e and n_w instances, respectively. (The results of BB and LS were provided by the authors. The results of LS are available only for the instance sets with 50 and 100 activities.) We should remark that the results of BB and LS are on a personal computer with 333MHz Pentium II processor with the computational time limit being 100 seconds for each instance, while our algorithm was run on a workstation with 3.0GHz processor with 10 seconds of the time limit. According to the authors, the local search algorithm LS has not been polished, because the main purpose of its implementation was to validate the idea of its neighborhood structure proposed in Neumann et al (2003).

Although a precise comparison is difficult because different processors were used, our algorithm is competitive to BB for instances with 10 and 20 activities, if we disregard the advantage of BB that is able to prove optimality. For instances with 50 and 100 activities, our algorithm found better solutions than BB and LS for many instances.

9.5.1.1 Effectiveness of the shift operation. To investigate the effectiveness of the shift operation explained in Section 9.4.5, we also solved the above instances with $|I| \geq 50$ by our algorithm without shift operations; i.e., Step 3 is skipped in procedure ITERATEDLOCALSEARCH. Table 9.2 shows the performance of ITERATEDLOCALSEARCH compared with the one not using shift operations, in the same manner as in Table 9.1. For some instances, the results are better if we do not use shift operations, but on the whole, the performance is improved by incorporating the shift operations.

9.5.2 Activity-crashing

The RCPSP/conv features variable activity durations. To see the effect of this flexibility, we solved instances prepared as follows. The instances are based on those generated by ProGen/max, which are RCPSPs to minimize the project duration subject to temporal constraints. Each temporal constraint is given by

Table 9.3. Impacts of activity crashing

Instance set	$\|I\| = 10$	$\|I\| = 20$	$\|I\| = 50$	$\|I\| = 100$
Comparison with the optimum	8-65-0	15-51-0	3-24-6	0-13-11
#feasible with activity-crashing	6/17	8/20	8/17	2/12

an inequality on the start times of two activities, $x_j^s - x_i^s \geq \delta_{i^s j^s}$. If the right hand side value $\delta_{i^s j^s}$ equals to the duration of activity i, this temporal constraint represents the precedence relation between activities i and j. We first rewrite such a constraint by $x_j^s - x_i^c \geq 0$. Then, we allow activity i to be shortened by one time period, if and only if its duration d_i (specified by ProGen/max) is more than one; i.e., we set the minimum and maximum durations of activities i by $d_i^{\min} := \max\{d_i - 1, 1\}$ and $d_i^{\max} := d_i$, respectively. We define the objective function as the project duration plus the number of crashed activities, which can be described by a convex function of time-lag costs.

We used the same four instance sets as in the previous subsection. In this experiment, we solved only those instances whose optimal values (minimum project durations) are known; i.e., 73, 66, 33 and 24 instances for 10, 20, 50 and 100 activities, respectively. We set the time limit for each run to $|I|/10$ seconds, where $|I|$ = 10, 20, 50 or 100 is the number of activities.

In real situations, if a problem turns out to be infeasible, it is often required to modify the problem by relaxing some conditions and/or constraints so that the problem becomes feasible. To demonstrate that activity-crashing is sometimes useful for this purpose, we also solved infeasible instances (17, 20, 17 and 12 instances in the four instance sets, respectively) by allowing shortening activity durations by at most one time period per activity. (Note that there is no guarantee that this modification makes such instances feasible.) In this experiment, the objective function is defined as the number of crashed activities.

Table 9.3 summarizes the results. The second row shows the number of instances for which our results are better than, equivalent to and worse than the optimal value in the same manner as in Table 9.1. (If the solution obtained by our algorithm is better than the optimum, it means that we succeeded in shortening the minimum project duration by T time periods by crashing at most $T - 1$ activities.) The last row shows the number of infeasible instances for which a feasible schedule can be found by allowing activity-crashing.

From this result, we can confirm the effect of activity-crashing, particularly for instance sets with $|I| \leq 20$. On the other hand, for some instances with $|I| \geq 50$, we failed to find a solution whose objective value is the same as the optimum, even though problems are relaxed. This observation indicates that there remains a need to improve the performance of our algorithm.

Table 9.4. Impacts of relaxing temporal constraints

Instance set	$\|I\| = 10$	$\|I\| = 20$	$\|I\| = 50$	$\|I\| = 100$
Comparison with the optimum	6-67-0	12-51-3	2-26-5	4-13-7
#feasible with constraint relaxation	7/17	13/20	6/17	1/12

9.5.3 Relaxation of temporal constraints

As another application of the RCPSP/conv, we consider the RCPSP with soft temporal constraints. As in the experiment for activity-crashing in the previous subsection, we prepared problem instances by modifying those generated by ProGen/max. In this experiment, we relaxed temporal constraints imposed on two different activities i and j, if their durations are both greater than zero; i.e., i and j are not fictitious, but real activities. For each of such temporal constraints $x_v - x_u \geq \delta_{uv}$, we replace it with a slightly relaxed constraint $x_v - x_u \geq \delta_{uv} - 1$, and introduce a time-lag cost with $f_{uv}(\delta) := \max\{\delta_{uv} - \delta, 0\}$. This transformation allows us to violate temporal constraints by one time period with being charged a penalty. The objective function is the sum of the project duration and the number of violations of the soft temporal constraints. Also in this experiment, we solved the infeasible instances by defining the objective function as the number of violations of the soft temporal constraints. Other experimental settings are the same as in the previous subsection.

The results are shown in Table 9.4. We observed that the project duration can be reduced (sometimes to a large extent) by violating a smaller number of temporal constraints.

9.6 Conclusion

In this paper, we have introduced a generalized model of the RCPSP and proposed a metaheuristic algorithm. By incorporating variable activity durations and time-lag costs, it becomes possible to deal with complicated project scheduling problems in real applications. Since the proposed algorithm solves the minimum cost flow problem many times in its execution, from viewpoint of computation time, it may not be practical to solve instances of very large sizes. From the experimental results, however, we can observe that the proposed algorithm performs well for instances with up to around 100 activities.

Acknowledgments

We appreciate Professor C. Schwindt of University of Karlsruhe for providing us with the benchmark instances and the computational results used in Section 9.5. We are also grateful to Professor A. Frangioni of University of

Pisa for providing us with the source code of RelaxIV. This research was partially supported by Grants-in-Aid for Young Scientists and Grant-in-Aid for Scientific Research on Priority Areas "New Horizons in Computing" from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

Ahuja, R.K., Magnanti, T.L. and Orlin, J.B. (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall.

Ahuja, R.K., Hochbaum, D.S. and Orlin, J.B. (1999). Solving the convex cost integer dual network flow problem. in: *Lecture Notes in Computer Science* 1610 (IPCO'99), G. Cornuejols, R.E. Burkard and G.J. Woeginger, eds, pp. 31–44.

Bartusch, M., Möhring, R.H. and Radermacher, F.J. (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research* 16:201–240.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* 112:3–41.

Bertsekas, D.P. and Tseng, P. (1994). RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems. Laboratory for Information and Decision Systems Report P-2276, Massachusetts Institute of Technology, Cambridge.

Cherkassky, B.V. and Goldberg, A.V. (1997). On implementing push-relabel method for the maximum flow problem. *Algorithmica* 19:390–410.

Demeulemeester, E.L. and Herroelen, W.S. (2002). *Project Scheduling: A Research Handbook*, Kluwer, Boston.

De Reyck, B. and Herroelen, W. (1998). A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Reserch* 111:152–174.

Herroelen, W., De Reyck, B. and Demeulemeester, E.L. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers and Operations Research* 25:279–302.

Klein, R. (2000). *Scheduling of Resource-Constrained Projects*, Kluwer, Boston.

McCormick, S.T. and Ervolina, T.R. (1994). Computing maximum mean cuts. *Discrete Applied Mathematics* 52:53–70.

Neumann, K., Schwindt, C. and Zimmermann, J. (2002). *Project Scheduling with Time Windows and Scarce Resources*, Springer, Berlin.

Neumann, K., Schwindt, C. and Zimmermann, J. (2003). Order-based neighborhoods for project scheduling with nonregular objective functions, *European Journal of Operational Research* 149:325–343.

Schwindt C. (1998). Generation of resource-constrained project scheduling problems subject to temporal constraints. Report WIOR-543, Institute for Economic Theory and Operations Research, University of Karlsruhe.

Schwindt, C. (1999). Minimizing earliness-tardiness costs of resource-constrained projects. in: *Operations Research Proceedings*, K. Inderfurth, G. Schwoe-diauer, W. Domschke, F. Juhnke, P. Kleinschmidt and G. Waescher, eds, Springer, pp. 402–407.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W. (2001). An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research* 102:179–196.

Weglarz, J. (ed.) (1999). *Project scheduling: Recent Models, Algorithms, and Applications*, Kluwer, Boston.

Chapter 10

A HYBRID GENETIC ALGORITHM BASED ON INTELLIGENT ENCODING FOR PROJECT SCHEDULING

Javier Alcaraz and Concepción Maroto
Department of Applied Statistics, Operations Research and Quality
Universidad Politécnica de Valencia
jalcaraz@eio.upv.es

Abstract In the last few years several heuristic, metaheuristic and hybrid techniques have been developed to solve the Resource-Constrained Project Scheduling Problem (RCPSP). Most of them use the standard activity list representation, given that it seems to perform best in solving the RCPSP independently of the paradigm employed (genetic algorithms, tabu search, simulated annealing, ...). However, we have designed an innovative representation, one which has not been used before and which includes a lot of problem-specific knowledge. Based on that representation we have developed a new competitive and robust hybrid genetic algorithm, which uses genetic operators and an improvement mechanism specially designed to work on that representation and exploit, in a very efficient way, the information contained in it. We have compared this algorithm with the best algorithms published so far, using the standard benchmark of PSPLIB. The results show the excellent performance of our algorithm.

Keywords: Project Scheduling, Genetic Algorithms, Hybrid Algorithms, Metaheuristic Techniques.

10.1 Introduction

The resource-constrained project scheduling problem (RCPSP) is a NP-hard optimization problem which has been widely studied in the literature. Since the first works developed to solve it in the 60s, several different optimization techniques have been proposed to solve this problem. Following the categorization of Kolisch and Hartmann (1999) these techniques can be divided into exact methods, heuristics, classical metaheuristics, non-standard metaheuristics and other techniques. The last heuristic methods, including metahuristics, are described and compared by Hartmann and Kolisch (2000) and Kolisch and Hartmann (to appear). One of the most important aspects to consider when designing an algorithm to solve this problem is how to encode the solutions. A study carried out by Hartmann (1998) determined that the activity list representation (ALR) performs the best to solve the RCPSP, when compared with other encodings, irrespective of the paradigm employed. So, the ALR and some of its extensions are the most commonly used encodings in the algorithms designed to solve the RCPSP although in the literature we can find some others which have obtained good results. In this paper we propose a hybrid genetic algorithm based on a new efficient encoding for the solutions, which incorporates problem-specific knowledge. This representation is an extension of the ALR and combines the features of two encodings previously used by Hartmann (1999) and Alcaraz and Maroto (2001). We have also extended the genetic operators designed by the aforementioned authors to work up on this encoding and use, in a very efficient way, the problem-specific information contained in it. Moreover, we have developed two different improvement procedures; the first can be applied to the chromosomes in the population in order to improve their quality and the second permits us to introduce variability in the population and avoid being trapped in a local optimum.

The outline of the remaining is as follows: section 16.2 shows the problem formulation. In section 16.3 the new solution representation is described. The new crossover and mutation operators designed to work up on that encoding are presented in section 16.4 and section 16.5 describes two procedures designed to improve the performance of the genetic process. Section 16.5 presents the results of the extensive computational experience carried out in order to configure the algorithm and compare it with the best techniques that have appeared so far. Finally, we draw the main conclusions and examine directions of future research in section 15.6.

10.2 Problem formulation

The problem considered in this paper is the Non Pre-emptive Single Mode Resource-Constrained Project Scheduling Problem (RCPSP), in which the objective is to minimize the makespan or total project duration.

This problem is concerned with a set J of N activities, $J = \{1, \ldots, N\}$ and a set R of K renewable resources, $R = \{1, \ldots, K\}$. For each resource $k \in R$, we know its total amount or availability per-period which is constant and given by A_k. An activity j has to be processed for d_j time units. Preemption of activities is not allowed, that is to say, when an activity starts it must be executed period by period until it is completed (the d_j periods of execution time must be consecutive). In each period of its execution time $t = 1, \ldots d_j$, activity j requires r_{jk} units of resource k to be successfully executed, being $r_{jk} \Leftarrow A_k$.

Between some of the activities there are precedence relations of the finish-start type with a zero parameter value defined between the activities. Activity j can not start until all its immediate predecessors, given by the set P_j, have completely finished. The values of A_k, d_j and r_{jk} (availability of resources, duration of activities and requirements of resources by activities) are integer and non-negative.

The objective is to determine the starting time of each activity, so that the project makespan or total project duration is minimized, and both the precedence and the resource constraints are satisfied. A schedule can be presented as $S = \{S_1, \ldots, S_N\}$, where Sj denotes the start time of activity j.

This problem is noted as $m, 1|cpm|C_{max}$ in the notation proposed by Herroelen et al (1998) and $PS|prec|C_{max}$ in the one proposed by Brucker et al (1998).

10.3 Intelligent encoding

Different representations have been proposed in the literature to encode the solutions of the RCPSP. The standard activity list representation (ALR) is the most commonly used, given that, as a study carried out by Hartmann (1998) determined, the ALR or permutation based encoding is the best when solving this problem, with independence on the paradigm used (genetic algorithms, tabu search, simulated annealing...). Among the most recent algorithms which have used this standard representation we can mention the works of Bouleimen and Lecocq (2003), Fleszar and Hindi (2004), Hindi et al (2002), Jozefowska et al (2001), Klein (2000), Nonobe and Ibaraki (2002), Tormos and Lova (2003) and Zhang et al (2005). However, several authors have designed new representations, based on the ALR which improve its performance, adding problem-specific knowledge (Alcaraz and Maroto (2001), Hartmann (1998)). Other authors have employed different representations, which are not based on the ALR. Some of them such as Debels et al (2006), Kim et al (2005), Valls et al (2003), Valls et al (2005), Zhang et al (2005) have used the random key (or priority value) representation or even extensions of it. Hartmann (1998) and Özdamar (1999) use the priority rule representation in their genetic algorithms. Sampson and Weiss (1993) developed a variant of a simulated annealing algorithm employing the shift vector representation to encode the solutions. Other authors have recently used direct encodings, such as Toklu (2002) or Thomas and Salhi (1997), directly working on schedules. Other completely different encodings have been recently used by authors such as Artigues et al (2003).

In sections 16.3.1, 10.3.2 and 10.3.3 the ALR representation and two extensions from it are presented. In section 10.3.4 we describe the new representation designed, which is a combination of the ones presented in sections 10.3.2 and 10.3.3.

10.3.1 Activity list representation

In this kind of representation, the solution is encoded as a precedence feasible list of the activities. Each activity can appear in the list in any position after all its predecessors. To construct the related schedule we could apply the serial method proposed by Kelley (1963). We would schedule the activities, one by one, in the order given by the list, so when an activity is going to be scheduled, all its predecessors have already been scheduled (forward scheduling). Each activity is assigned the earliest feasible start time. It is interesting to point out that the parallel method could not be directly applied to this representation to transform the individual into its corresponding schedule.

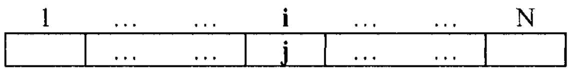


Figure 10.1. Activity List Representation

Each individual in the population is represented by an array with as many positions as activities in the project. In Figure 10.1, we can observe the activity list representation for a project with N activities. Activity j will be the i-th activity chosen to be scheduled. It will be scheduled in its earliest feasible start time. When activity j, located in position i, is chosen to be scheduled, all its predecessors, which will appear in some position $1, \dots, i-1$, will have already been scheduled. In this way, the related schedule will always be a feasible schedule. Notice that when applying this procedure, one and only one schedule (phenotype) can be deduced from a given sequence (genotype), but different sequences could transform into the same schedule. When applying the serial method to transform the representation into a schedule, the search space is formed by the set of active schedules, which always contains an optimal solution (Kolisch (1996), Sprecher et al (1995)).

10.3.2 Hartmann's extended representation

Computational studies carried out by Hartmann (1999) revealed that the activity list representation yields the most promising result in a genetic algorithm, when compared with other representations such as random key or priority rule encoding. He used the activity list representation and the serial schedule generation scheme (SGS) or serial method as decoding procedure. However, the author extends the activity list representation in order to allow both serial and parallel schedule generation schemes to be employed as decoding procedures. As the parallel scheduling method is not immediately applicable to deal with activity lists, the author adapts the activity selection mechanism in order to directly work up on activity lists. The extension of the representation is made by adding a new gene, a boolean indicator, which determines if the serial or the parallel SGS is employed to build up the schedule. As the author explains, this new representation not only determines the schedule itself, but also the algorithm with which it is constructed. The algorithm learns which scheduling method is the better choice and adapts itself accordingly. In this way, the algorithm is transformed into a self-adapting genetic algorithm, in which not only the solution of the problem but also the algorithmic structure is subject to genetic optimization. Computational experiments carried out by the author demonstrate the good performance of this encoding, when compared with the standard activity list representation.

10.3.3 Activity list with scheduling mode representation

This representation was designed by Alcaraz and Maroto (2001) and extended later by Alcaraz et al (2003) to deal with the multi-mode version of the RCPSP. It is based on the standard activity list representation described in section 16.3.1. Now a solution is represented by a pair: an activity list and an additional gene, called forward/backward (f/b) gene, which indicates the way in which the schedule is built. We can observe this representation in Figure 10.2. Given a (precedence feasible) list of activities, and applying the serial

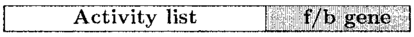


Figure 10.2. Activity List with Scheduling Mode Representation

method, there are two different ways of constructing the related schedule: forward scheduling and backward scheduling. In forward scheduling, when an activity is chosen to be scheduled, all its predecessors must have already been scheduled (the first activity chosen to be scheduled is the first of the list), and the activity is scheduled in its earliest feasible start time. In backward scheduling, when an activity is chosen to be scheduled, all its successors must have already been scheduled (the first activity chosen to be scheduled is the last one in the list), and it is scheduled in its latest feasible start time. For a detailed description of forward and backward scheduling readers are referred to Elmaghraby (1977).

The interest of this representation is to exploit the possibility of scheduling a solution in a forward/backward way, which produces left/right shifted schedules with possibly different makespan. Therefore, the same activity list, with different scheduling modes, may transform into different schedules, with possibly different makespan. As in the two previous representations, different solutions can transform into the same schedule, but an individual transforms into one, and only one schedule. In their study, the authors demonstrate its good performance.

10.3.4 New encoding

The new encoding combines the two representations described in subsections 10.3.2 and 10.3.3 in a very efficient way. The extended representation proposed by Hartmann allows the genetic process itself to decide the schedule generation scheme which is used to transform the activity list into a schedule: serial or parallel. Computational studies carried out by Hartmann and Kolisch (2000) demonstrated that the parallel method obtains better results than the serial one when applied to large projects, because the serial SGS searches in a larger solution space. However, in smaller projects where the solution space is also more reduced, this space could not contain the optimal solution, which is a drawback. So, depending on the project to be solved, the algorithm can learn and decide which SGS is better. This is a very good feature which will be incorporated in the new representation. On the other hand, the activity list with scheduling mode representation proposed by Alcaraz and Maroto allows the algorithm to employ forward or backward scheduling. By applying backward scheduling we can build schedules which can not be obtained by forward scheduling and vice-versa.

At this point, it is interesting to point out that with Hartmann's extended representation, the algorithm could employ the serial or the parallel SGS, but always applying forward scheduling. On the other hand, with the activity list with scheduling mode representation the algorithm could apply forward or backward scheduling, but always with the serial method. The backward-parallel schedule generation scheme could not be employed with any of these encodings to transform the activity list into a schedule. The purpose of the new representation is to make possible the four different scheduling alternatives: forward-serial, forward-parallel, backward-serial and backward-parallel. To this end, we combine the two previous encodings, adding to the standard activity list representation two new genes: the f/b gene used in the activity list with scheduling mode representation and the boolean indicator of the Hartmann's extended representation, called henceforth serial/parallel (s/p) gene. As the parallel scheduling generation scheme is not immediately applicable to deal with activity lists, when the forward (backward) parallel scheme is used, the position number of an activity will indicate its priority to be scheduled; the activity with the lowest (highest) position being the activity with the highest priority.

Activity list	f/b gene	s/b gene

Figure 10.3. New Encoding

This new encoding is represented in Figure 10.3. As we can see, for a project with N non-dummy activities, we have an activity list with N genes and two additional genes, in total $N + 2$ genes.

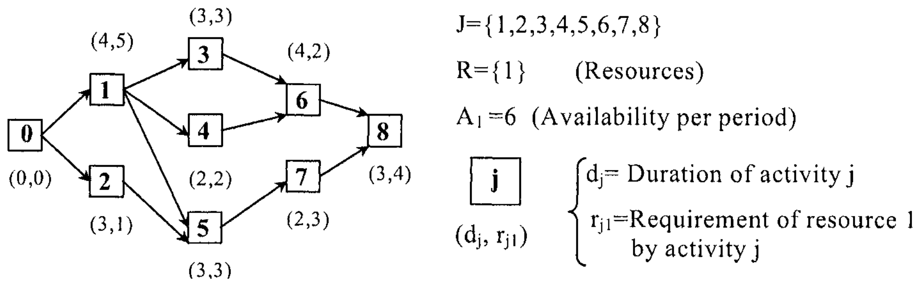


Figure 10.4. Project Example

In Figure 10.4 a project example with eight non-dummy activities has been represented. Only one resource is required for the activities. The daily availability of this resource is 6 units. The duration (in days) and the units per day required by each activity are also given in parentheses below the activities.

Solution A	1	2	5	4	3	6	7	8	f	→ Forward (serial)
Solution B	1	2	5	4	3	6	7	8	b	→ Backward (serial)
Solution C	1	2	5	4	3	6	7	8	s	→ Serial (forward)
Solution D	1	2	5	4	3	6	7	8	p	→ Parallel (forward)

Figure 10.5. Different Solutions. A and B: Activity list with scheduling mode representation. C and D: Hartmann's extended representation

In Figure 10.5 we have represented different solutions to the project instance given in Figure 10.4 using the encodings described in subsections 10.3.2 and

10.3.3. Solutions A and B use the activity list with scheduling mode representation and solutions C and D Hartmann's extended representation. The four solutions share the same activity list, but different f/b gene or s/p gene. Solution A will be transformed into a schedule applying the serial SGS in a forward way, and solution B with serial-backward SGS. On the other hand, solutions C and D will be transformed using forward scheduling, with the serial or parallel SGS respectively. The schedules obtained after applying the corresponding method are given in Figure 10.6. As we can see, solutions A, C and D transform into the same schedule, giving a makespan of sixteen days. Solution B has transformed into a different schedule, which gives, in this case, a lower makespan. These two schedules are completely different. However, this activity list could also be transformed into a different schedule if we apply the backward-parallel SGS, which is not possible with Hartmann's extended representation or with the Alcaraz and Maroto encoding.

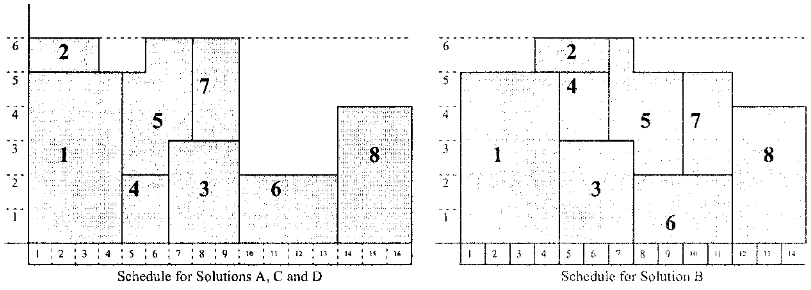


Figure 10.6. Schedules for solutions given in Figure 10.5

In Figure 10.7 a solution for the previous project instance employing the new encoding has been represented. This solution has the same activity list as solutions represented in Figure 10.5. The f/b gene indicates backward scheduling and the s/p gene is p, so the schedule will be built using the backward-parallel schedule generation scheme. This schedule is shown in Figure 10.8 and, as we can see, it is different from schedules given in Figure 10.6. This schedule could not have been obtained with the same activity list using any of the two previous representations.

Solution E	1	2	5	4	3	6	7	8	b	p	→ Backward-Parallel

Figure 10.7. Solution with the new encoding

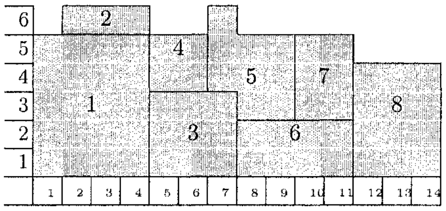


Figure 10.8. Schedule for Solution represented in Figure 10.7

The new representation we propose is intelligent in the sense that it permits the algorithm to decide which of the scheduling possibilities is the best, depending on the project characteristics, but without excluding any of the alternatives: forward-serial, forward-parallel, backward-serial and backward-parallel. Some activity list will be transformed in a better schedule using one method or the other, but the most adequate method does not need to be stated before applying the algorithm. The genetic process adapts during its execution and learns which of the alternatives perform best for the instance to be solved.

In the next section we describe the genetic operators designed to work up on this new representation and use, in a very efficient way, the problem-specific knowledge which is included in it.

10.4 Genetic operators

The performance of a genetic algorithm greatly depends on how the genetic operators have been designed. Moreover, the crossover mechanism is the operator with the higher influence in the genetic process, as was demonstrated by Alcaraz (2001). We have adapted the crossover and mutation operators previously designed by Alcaraz and Maroto (2001), which reported excellent results, to work over the new representation, in order to use the problem-specific knowledge contained in it in an efficient way.

10.4.1 Extended crossover

The crossover process combines the features of two parent chromosomes to form two offspring which inherit their characteristics. A poorly designed crossover operator becomes a sort of mutation. In the crossover process, individuals in the populations are mated randomly and each pair undergoes the operation with a given probability, creating two children by crossover. The parent population is replaced by the offspring population. If parents do not undergo the crossover operation, they remain unaltered in the next generation. So, the population size is always the same.

We have extended the two-point forward-backward crossover operator designed by Alcaraz and Maroto (2001) in order to manage the new representation presented in subsection 10.3.4. Now, the children inherit the f/b gene and the s/p gene from their respective progenitor: the son from the father and the daughter from the mother. The parents' f/b genes determine the way in which the crossover operation is carried out. The mother's f/b gene indicates the way in which the daughter's list is built and the father's f/b gene directs the generation of the son's. In this way, the problem-specific information which is given in the encoding is used in the crossover process to generate better individuals. The parents' s/p genes do not influence the way the offspring are generated, although these are inherited by the children.

First of all, two integer and non-negative crossover points $k1$ and $k2$ are randomly generated ($k1 < k2$). The offspring activity lists are generated as follows. If the progenitor's f/b gene indicates *forward* the corresponding child directly inherits the first $k1$ positions of the progenitor's list, in the same order. The positions between $k1 + 1$ and $k2$ are taken preserving their relative order in the other progenitor and the positions between $k2 + 1$ and N are again taken from the first progenitor, preserving their relative order. The process changes if the progenitor scheduling mode indicates *backward*. Now, the last positions are directly inherited in the activity list, the central positions are taken from the other progenitor, preserving the relative order among the activities, and the first genes, again form the initial progenitor, with their relative order.

An example of this operation is given in Figure 10.9. We have eight possible progenitors, the mother and the father presented in the figure combining the f/b and the s/p genes. The two random crossover points generated are positions two and six, which divide each activity list into three segments. If the mother is forward, the daughter created is Daughter A, who inherits the f/b gene from her. As the daughter also inherits the s/p gene from the mother, we could have Daughter A with serial or parallel SGS. The two first positions in the daughter's list are directly taken from the mother, activities 1 and 3. The four following positions are inherited from the father, preserving the relative order among the four activities in the father. So, the first four activities in the father which still do not appear in the daughter are, in this order, activities 2, 4, 5 and 7. The last two positions are again taken from the mother, preserving again the relative order between these two activities. The two activities which still do not appear in the daughter are activities 6 and 8, which present this order in the mother and must preserve this relative order in the daughter. As we can see, the father's f/b and s/p genes do not influence the generation of the daughter, but they direct the creation of the son. Let us suppose now that the father is backward. The son generated will be Son 2, and the s/p gene will be the same as the father's. The generation of the son's activity list will be from the end to the beginning. The two last positions in the activity list are directly inherited from the father, activities 8 and 6. The following activity will be the one which still does not appear in the son and presents a higher position number in the mother, activity 7. The three following positions are taken from the mother following this reasoning. To finish, positions two and one are again inherited from the father, activity 3 has a higher position number than activity 1, so this is the relative order between them.

Employing this extended operator, with two parents' activity lists, combining the f/b and the s/p genes, we could have up to sixteen different crossover combinations. This crossover operator permits all the possible combinations, so that the genetic process itself decides which is the most appropriate schedule generation scheme, without excluding a priori any of them. The computational results given in section 10.6.1 show that the use of this representation outperforms the results given by the other encodings presented.

10.4.2 Extended mutation

After the parent population has been replaced by the offspring population, the mutation operator is applied to the latter. Mutation alters one or more genes of a selected solution to reintroduce lost genetic material and introduce some extra variability into the population.

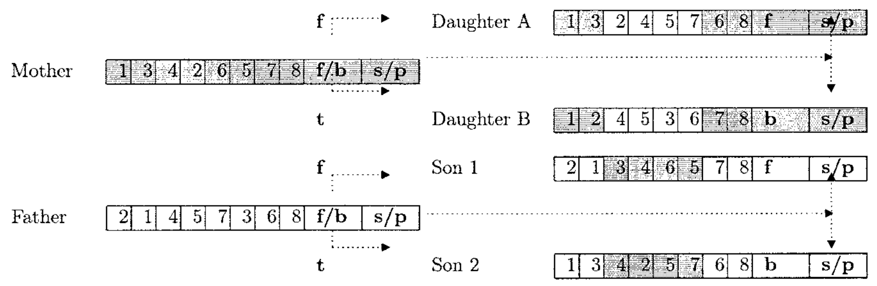
We have extended the mutation mechanism designed by Alcaraz and Maroto (2001) in order to work over the new representation. This extension implies that the s/p gene could also be altered by the mutation operator. That is, the s/p gene could transform from *serial* to *parallel* or from *parallel* to *serial* with a given probability, P_m. The mutation operator operates in two phases. First of all, the activity list is mutated. Later, the mutation process is applied to the f/b and the s/p genes. Each position of the list is mutated with a given probability of P_m.

In the first phase, for each activity in the sequence, a new position is randomly chosen, between the highest position of its predecessors and the lowest position of its successors so that only precedence feasible solutions are generated. The activity is inserted into the new position with a probability of P_m.

In the second phase, the f/b and s/p genes could be altered with the same probability P_m. That is, the f/b gene could change from f to b or from b to f with this probability. In the same way, the s/p gene could be transformed with a probability of P_m. We would like to remark that by only changing one of these genes, the corresponding schedule could change completely, that is, the same activity list could transform into a completely different schedule.

In Figure 10.10, an example of this operation is shown. In the first phase, the operator reviews the activity list, and each activity is replaced in a different position with a given probability P_m. In this example, the first phase of the mutation operation only alters two genes. First of all, activity 3, which is in the

Random crossover points:
$q_1 = 2; q_2 = 6$



Parents crossed		Offspring	Parents crossed		Offspring
Mother;f;s – Father;f;s	$\longrightarrow$	Daughter A;s – Son 1;s	Mother;b;s – Father;f;s	$\longrightarrow$	Daughter B;s – Son 1;s
Mother;f;s – Father;f;p	$\longrightarrow$	Daughter A;s – Son 1;p	Mother;b;s – Father;f;p	$\longrightarrow$	Daughter B;s – Son 1;p
Mother;f;p – Father;f;s	$\longrightarrow$	Daughter A;p – Son 1;s	Mother;b;p – Father;f;s	$\longrightarrow$	Daughter B;p – Son 1;s
Mother;f;p – Father;f;p	$\longrightarrow$	Daughter A;p – Son 1;p	Mother;b;p – Father;f;p	$\longrightarrow$	Daughter B;p – Son 1;p
Mother;f;s – Father;b;s	$\longrightarrow$	Daughter A;s – Son 2;s	Mother;b;s – Father;b;s	$\longrightarrow$	Daughter B;s – Son 2;s
Mother;f;s – Father;b;p	$\longrightarrow$	Daughter A;s – Son 2;p	Mother;b;s – Father;b;p	$\longrightarrow$	Daughter B;s – Son 2;p
Mother;f;p – Father;b;s	$\longrightarrow$	Daughter A;p – Son 2;s	Mother;b;p – Father;b;s	$\longrightarrow$	Daughter B;p – Son 2;s
Mother;f;p – Father;b;p	$\longrightarrow$	Daughter A;p – Son 2;p	Mother;b;p – Father;b;p	$\longrightarrow$	Daughter B;p – Son 2;p

Figure 10.9. Extended Two-Point Forward-Backward Crossover Example

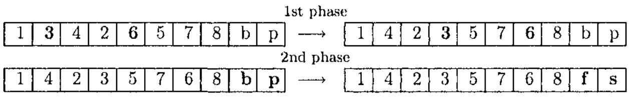

1st phase
1 3 4 2 6 5 7 8 b p → 1 4 2 3 5 7 6 8 b p
2nd phase
1 4 2 3 5 7 6 8 b p → 1 4 2 3 5 7 6 8 f s

Figure 10.10. Extended Mutation Operator Example

second position, is inserted into a new position, position number four, so that the precedence relations constraints are satisfied. Then, activity 6, which is placed in the fifth position, is moved to position seven. After these two replacements, the sequence is a precedence feasible list. In the second phase, the operator can mutate the f/b and s/p genes with the same probability. In this example, both, f/b and s/p genes are altered, changing from b to f and from p to s respectively.

10.5 Improving the performance of the algorithm

10.5.1 Local search procedure

We have hybridized the algorithm incorporating a local search procedure (LSP) in order to improve the quality of solutions. After the crossover and mutation operations, the LSP can be performed on the current population. The procedure is based on the mechanisms previously designed and efficiently used by different authors such us Tormos and Lova (2001) or Valls et al (2005). Activities are sorted depending on their start/finish time in the schedule. Then, this sorted sequence replaces the original activity list. This sorting procedure can be applied in one or two phases. In our procedure, the problem-specific knowledge incorporated into the solution is exploited again in an efficient way. The procedure carried out depends on the f/b gene of the solution.

In the first phase, the solution is transformed into its schedule, applying the corresponding serial/parallel forward/backward schedule generation scheme. Then, activities are sorted with respect to non-decreasing finish times if the f/b gene indicates f and with respect to non-decreasing start times if it is b. Then, this reordered list replaces the original one and the f/b gene is changed, from forward to backward or vice-versa. The effect of this reordering is to right (left) shift the schedule if the serial SGS is applied. Thus, in this case this procedure can not lead to an increasing of the schedule makespan. However, if the parallel SGS is being applied, the LSP could worsen the quality of the solution. In this case, the new solution does not replace the initial one.

The second phase, if applied, consists of repeating the procedure carried out in the first phase, taking into account that the f/b gene has been changed at the end of this phase. If in the first phase the effect of this reordering has been to right shift the schedule, the effect of the second phase will be to left shift it, and vice-versa. It is possible that both phases lead to a decreasing of the project makespan, or only one of them or none of the phases.

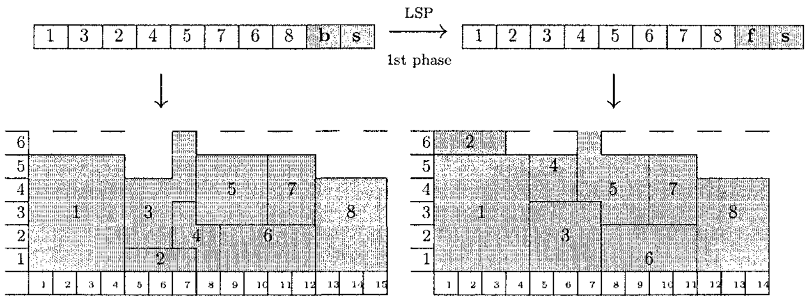


Figure 10.11. Example of application of the local search procedure

An example of the application of the first phase of the local search procedure described above has been represented in Figure 10.11. The initial solution has been transformed into its corresponding schedule, applying the backward-serial schedule generation scheme, giving a makespan of 15 time units. Then, activities are sorted with respect to non-decreasing start times, the f/b gene has been changed to f and the schedule has been built up again employing now the forward-serial SGS. As we can see, the project makespan has been reduced to 14 time units. Now, the second phase could be applied to this new solution.

10.5.2 Random replacement procedure

We have also included in the algorithm the random replacement procedure (RRP) designed by Alcaraz et al (2003) to deal with the multi-mode version of the problem. This operation is applied to a given population with a probability of P_{replac}. If it is performed in the current generation, each individual of the population is exchanged by a random generated solution with a probability of P_{exch}. A random solution consists of a solution where the activity list has been generated preserving the precedence feasibility and the f/b and the s/p genes have been randomly generated. The RRP allows us to restart the population or reintroduce some variability when it has prematurely converged or has been trapped in a local optimum. However, we use an elitist model; therefore, the best individual in the current population always survives in the next generation.

10.6 Computational results

We have carried out an extensive computational experiment which has been developed in two phases. First, we have calibrated the algorithm, testing the performance of the new encoding, the extended genetic operators developed and the new procedures incorporated into the genetic algorithm. Then, in the second phase we have compared our hybrid algorithm with some of the best metaheuristics that have appeared in the literature so far. In the two following subsections we describe the computational experiments carried out. In our experiments, we have used the three standard test sets J30, J60 and J120 from the project scheduling problem library PSPLIB. For a more detailed description of these instance sets, we refer to Kolisch and Sprecher (1996). We have also used the number of schedules as termination criteria, computing 1,000, 5,000 and 50,000 schedules. After each execution of the algorithm, we have measured the average deviation from the optimal makespan for set J30 and the average deviation from the critical path-based lower bound when solving the projects with 60 or 120 activities.

10.6.1 Callibration of the algorithm

Calibrating or configuring the algorithm consists of setting the genetic operators (selection, crossover and mutation), parameters (population size, crossover probability, mutation probability...), improving procedures and encoding in order to get the best performance. There is not combination which always gives the best results, irrespective of the project characteristics or the termination criteria. However, we have based ourselves on preliminary experiments (Alcaraz (2001)) to select some of the operators and parameters. Some of them do not greatly influence the behaviour of the algorithm. We have selected the 2-tournament as selection procedure. That is, two individuals are randomly chosen from the population and compete for survival. Only the best of them will appear in the following generation. This procedure is repeated until the new population is completed. For a more detailed description of this and other selection mechanisms, readers are referred to Goldberg (1989). The crossover probability has been fixed at 80%. The mutation probability depends on the project size. When computing small projects (30 activities) the mutation probability is fixed at 5%, but it is decreased to 1% when the project size increases (60 or 120 activities). The population size also varies depending on the termination criterion; or rather the number of schedules computed in our case. So, when the number of computed schedules is 1,000, the population size used is of 50 individuals. For a larger number of computed schedules the population has a total of 100 individuals.

The first step is to analyze the performance of the new encoding as well as the crossover and mutation mechanisms designed to work up on it. To do that, we have compared the algorithm developed by Alcaraz and Maroto (2001), AM01, which makes use of the activity list with scheduling mode representation, the two-point forward backward crossover and the mutation procedure, with the new algorithm, which employs the new encoding and the extended crossover and mutation operators specifically designed to work over it. That is, the only differences between the algorithms are the encoding and the crossover and mutation operators. The rest of the parameters are identical. We have solved sets J30, J60 and J120 computing 1,000, 5,000 and 50,000 schedules with both algorithms. Table 16.1 summarizes these results.

As we can see in Table 16.1 the use of the new encoding and operators outperforms the results in all the test sets analyzed irrespective of the number of schedules computed. We can also observe that the algorithm does not prematurely converge when computing 1,000 or 5,000 schedules because of increasing the number of computed schedules, from 1,000 to 5,000 and from 5,000 to 50,000 the deviation from optimum or critical path-based lower bound decreases.

Table 10.1. Performance of the new encoding and genetic operators

	Max. Schedules					
	1,000		5,000		50,000	
Set	Encoding and genetic operators					
	AM01	New	AM01	New	AM01	New
J30	0.373	**0.334**	0.174	**0.131**	0.113	**0.083**
J60	12.62	**12.41**	12.05	**11.83**	11.70	**11.51**
J120	39.36	**37.17**	36.67	**35.44**	34.49	**34.11**

The second step consists of analyzing the performance of the algorithm when applying the local search procedure described in section 10.5.1. This procedure can be applied in one or two phases. Moreover, we must decide when to apply the procedure, that is, in which generations it is carried out. We have compared the application of the LSP to all the generations (All) or only to the schedules of the last generation (Last). In each case, we have applied one and two phases of the mechanism. The results are summarized in Table 16.2.

The results of this computational experiment show that this procedure improves the quality of the solutions if we compare these results with those presented in Table 16.1. For set J30, we have reduced the deviation from the optimal makespan, from 0.334, 0.131 and 0.083 to 0.157, 0.087 and 0.022 respectively when computing 1,000, 5,000 and 50,000 schedules. For set J60 the deviation from the critical path-based lower bound has decreased from 12.41, 11.83 and 11.51 to 11.67, 11.20 and 10.93 respectively. Finally, for the projects with 120 activities, we have reduced the deviation from 37.17 to 34.97 when computing 1,000 schedules, from 35.44 to 33.51 when building 5,000 schedules and from 34.11 to 31.38 when the maximum number of computed schedules is 50,000. So, we can conclude that this local search procedure is an efficient mechanism

Table 10.2. Performance of the LSP

	Max. Schedules											
	1,000				5,000				50,000			
	Generation											
	Last		All		Last		All		Last		All	
Set	Number of phases											
	1	2	1	2	1	2	1	2	1	2	1	2
J30	0.297	0.314	**0.157**	0.216	0.158	0.167	0.090	**0.087**	0.098	0.108	**0.022**	0.088
J60	12.28	12.30	11.69	**11.67**	11.80	11.85	**11.20**	11.27	11.61	11.58	10.98	**10.93**
J120	36.63	36.25	**34.97**	**34.97**	35.22	35.19	**33.51**	33.69	34.16	34.11	31.81	**31.38**

to be incorporated into the genetic algorithm. If we analyze the behaviour of the procedure, we can deduce that it is always better to apply the procedure to all the generations than to only the final one, although each application of the procedure implies the computation of one or two schedules, depending on the phases carried out. So, applying the procedure to all the generations implies reducing the number of generations because in each generation the number of computed schedules is much higher. With respect to the application of one or two phases, there do don't seem to be great differences. In order to generalize the configuration of the LSP irrespective of the number of schedules computed and the project instances solved, we will configure the algorithm incorporating the LSP applied to all the generations in only one phase.

The final step of this configuration process consists of analyzing the performance of the random replacement procedure. Two parameters must be set in order to configure this mechanism, P_{replac} and P_{exch}. We have established two different levels for the replacement probability, 5% and 15%, and three levels for P_{exch}, 50%, 70% and 90%. The results are shown in Tables 16.3, 16.4 and 16.5.

Table 10.3. Performance of the RRP. J30 set

	Max. Schedules					
	1,000		5,000		50,000	
P_{exch}	P_{replac}					
	5%	15%	5%	15%	5%	15%
50%	**0.148**	0.164	0.082	0.076	0.018	0.019
70%	0.181	0.188	0.071	**0.058**	0.016	0.013
90%	0.189	0.161	0.070	0.063	**0.007**	0.011

Table 16.3 presents the deviation from the optimal makespan for the instances with 30 activities, when computing 1,000, 5,000 and 50,000 schedules combining the three exchange and the two replacement probabilities. If we compare these results with those shown in Table 16.2, where no replacement procedure has been carried out, we can deduce that this procedure has managed to decrease the deviation irrespective of the number of schedules constructed. The deviation has decreased from 0.157 to 0.148 when the maximum number of computed schedules is 1,000, from 0.087 to 0.058 for 5,000 schedules and from 0.022 to 0.007 when 50,000 schedules are generated. The J30 set is the easiest one to solve, and the algorithm can converge or be trapped in a local optimum in the majority of cases in a reduced number of generations. In those cases, this algorithm reintroduces variability into the population and is able to escape from that local optimum, improving the results obtained.

Table 10.4. Performance of the RRP. J60 set

	Max. Schedules					
	1,000		5,000		50,000	
P_{exch}	P_{replac}					
	5%	15%	5%	15%	5%	15%
50%	11.74	11.75	**11.05**	11.25	10.89	10.88
70%	**11.71**	11.80	11.22	11.29	10.86	**10.80**
90%	11.79	11.81	11.21	11.32	10.82	10.90

The RRP is also beneficial when solving the projects with 60 activities when the number of schedules is medium or high, decreasing the deviation from the critical path-based lower bound. When the maximal number of computed schedules is 1,000, the procedure is not able to reduce the deviation. This is due to the fact that set J60 is more difficult than set J30, and most of the instances need more than 1,000 schedules to converge. When computing 5,000 schedules the deviation has been reduced from 11.2 to 11.05, and from 10.93 to 10.80 when 50,000 schedules are built.

Table 16.5 displays the results for the most difficult set of instances solved, set J120, where the deviation from the critical path-based lower bound has been measured. These instances, with a much larger number of activities, need many more generations to converge, and the extra variability introduced by this random algorithm is not beneficial. As we can see, the results worsen if compared with those presented in Table 16.2, when this algorithm is not applied.

Therefore, we can conclude that the employment of the random replacement procedure is beneficial when solving sets J30 and J60 irrespective of the number of schedules computed, but is not advisable when the largest projects are

Table 10.5. Performance of the RRP. J120 set

	Max. Schedules					
	1,000		5,000		50,000	
P_{exch}	P_{replac}					
	5%	15%	5%	15%	5%	15%
50%	**34.97**	35.06	35.06	**33.64**	**31.81**	32.05
70%	34.98	35.14	**33.64**	33.91	31.94	32.23
90%	35.06	35.29	33.74	34.20	31.94	32.60

solved. The random nature of this algorithm does not permit us to establish a configuration which always performs better, although there are no great differences between the different alternatives for a given set and a specified number of schedules computed.

10.6.2 Comparison with other algorithms

We have compared our hybrid genetic algorithm with the best procedures that have appeared so far. We have compared with the works published in technical journals, books or proceedings of international conferences. Unpublished technical reports have not been considered. We have selected a variety of algorithms which are based on different paradigms, employ different representations or include different improvement mechanisms. These algorithms are, in alphabetical order:

- Alcaraz and Maroto (2001), Genetic algorithm.
- Bouleimen and Lecocq (2003), Simulated annealing.
- Debels et al (2006), Scatter search.
- Hartmann (2002), Genetic algorithm.
- Kochetov and Stoylar (2003), Genetic algorithm, Tabu search.
- Nonobe and Ibaraki (2002), Tabu search.
- Tormos and Lova (2003), Sampling.
- Valls et al (2005), Genetic algorithm.
- Valls et al (2005), Sampling.

The results given by these algorithms have been extracted from the work of Kolisch and Hartmann (to appear), in which they compared a great variety of algorithms for RCPSP. Therefore, to obtain these results, the authors managed to calibrate their algorithms and execute them with the best configurations. Tables 10.6, 10.7 and 10.8 show the results of this comparison. We have indicated the author of the algorithm, year of publication and type of algorithm: genetic algorithm (GA), tabu search (TS), scatter search (SS), simulated annealing (SA) or sampling (S). We have established as stopping criteria the maximum of 1,000, 5,000 and 50,000 schedules respectively. We can assume that the computational effort for building one schedule is similar in these heuristics. The algorithms have been sorted with respect to increasing deviation when computing 1,000 schedules. In the case of ties, the results for 5,000 schedules are used.

Table 10.6. Comparison of heuristics. Average deviation from optimal makespan. J30 PSPLIB

Author (Year)	Algorithm	Max Schedules		
		1,000	5,000	50,000
Kochetov and Stoylar (2003)	GA, TS	**0.10**	**0.04**	**0.00**
Alcaraz and Maroto (This work)	GA	0.15	0.06	0.01
Debels et al (2006)	SS	0.27	0.11	0.01
Tormos and Lova (2003)	S	0.30	0.16	0.07
Alcaraz and Maroto (2001)	GA	0.33	0.12	0.10
Valls et al (2005)	GA	0.34	0.20	0.02
Hartmann (2002)	GA	0.38	0.22	0.08
Bouleimen and Lecocq (2003)	SA	0.38	0.23	–
Nonobe and Ibaraki (2002)	TS	0.46	0.16	0.05
Valls et al (2005)	S	0.46	0.28	0.11

Results presented in Table 10.6 show that the best algorithm when solving small projects (30 activities) is the algorithm of Kochetov and Stoylar, irrespective of the number of schedules computed. The differences between this algorithm and the genetic algorithm presented in this work are rather small. The performance of the scatter search procedure of Debels et al is similar to previous algorithms when computing a large number of schedules, but rather worse when the maximum number of constructed schedules is smaller.

For the set of projects of medium size, 60 activities, the performance of the three algorithms, Debels et al, Kochetov and Stoylar and Alcaraz and Maroto is similar. The GA of this work performs better when computing 1,000 or 5,000 schedules, but a bit worse when the number of schedules is 50,000. The differences between these three algorithms and the rest are bigger.

For the largest projects, those with 120 activities, the situation is similar to the previous two. Now the best algorithm when computing 1,000 schedules is the procedure of Kochetov and Stoylar, improved by the scatter search procedure for 5,000 schedules. When solving the maximum number of schedules, 50,000,

Table 10.7. Comparison of heuristics. Average deviation (%) from critical path lower bound. J60 PSPLIB

Author (Year)	Algorithm	Max Schedules		
		1,000	5,000	50,000
Alcaraz and Maroto (This work)	GA	**11.67**	**11.05**	10.80
Kochetov and Stoylar (2003)	GA, TS	11.71	11.17	10.74
Debels et al (2006)	SS	11.73	11.10	**10.71**
Tormos and Lova (2003)	S	12.14	11.82	11.47
Valls et al (2005)	GA	12.21	11.27	10.74
Hartmann (2002)	GA	12.21	11.70	11.21
Alcaraz and Maroto (2001)	GA	12.57	11.86	11.70
Valls et al (2005)	S	12.73	12.35	11.94
Bouleimen and Lecocq (2003)	SA	12.75	11.90	–
Nonobe and Ibaraki (2002)	TS	12.97	12.18	11.58

the algorithm with the best performance is the new genetic algorithm of Alcaraz and Maroto.

We would like to highlight the great differences between our previous genetic algorithm (Alcaraz and Maroto (2001) and the new one, produced by the incorporation of the improvement mechanisms and the genetic operators which work up on the newly designed representation.

Table 10.8. Comparison of heuristics. Average deviation from critical path lower bound. J120 PSPLIB

Author (Year)	Algorithm	Max Schedules		
		1,000	5,000	50,000
Kochetov and Stoylar (2003)	GA, TS	**34.74**	33.36	32.06
Alcaraz and Maroto (This work)	GA	34.97	33.51	**31.38**
Debels et al (2006)	SS	35.22	**33.10**	31.57
Valls et al (2005)	GA	35.39	33.24	31.58
Tormos and Lova (2003)	S	36.24	35.56	34.77
Hartmann (2002)	GA	37.19	35.39	33.21
Valls et al (2005)	S	38.21	37.47	36.46
Alcaraz and Maroto (2001)	GA	39.36	36.57	34.40
Nonobe and Ibaraki (2002)	TS	40.86	37.88	35.85
Bouleimen and Lecocq (2003)	SA	42.81	37.68	–

Following Kolisch and Hartmann (to appear), to determine the best algorithms we use the concept of dominance. A heuristic a is dominated by a heuristic b if a has for at least one combination of instance set and number of generated schedules a higher average deviation than b without having for any of the other combinations a lower average deviation.

Table 10.9. Dominance of heuristics.

#Alg	Author (Year)	Algorithm	Dominated by #Alg
1	Alcaraz and Maroto (This work)	GA	–
2	Alcaraz and Maroto (2001)	GA	1, 4, 6
3	Bouleimen and Lecocq (2003)	SA	1, 2, 4, 5, 6, 8, 9, 10
4	Debels et al (2006)	SS	–
5	Hartmann (2002)	GA	1, 4, 6
6	Kochetov and Stoylar (2003)	GA, TS	–
7	Nonobe and Ibaraki (2002)	TS	1, 4, 6
8	Tormos and Lova (2003)	S	1, 4, 6
9	Valls et al (2005)	GA	4
10	Valls et al (2005)	S	1, 4, 5, 6, 8, 9

In Table 10.9, the ten algorithms compared have been sorted into alphabetical order. Then, for each algorithm, we have represented the algorithms that dominate it. As we can see, the only three heuristics which are not dominated by any other algorithm are those of Kochetov and Stoylar (2003), Debels et al (2006) and the genetic algorithm of Alcaraz and Maroto (this work). Therefore, these three algorithms are the best procedures in the comparison we have carried out.

10.7 Conclusions

In this paper we have presented a new encoding for the solutions of the RCPSP. It is an innovative representation, which includes two features of previous encodings: the possibility of employing the serial or the parallel SGS, and the combination of forward and backward scheduling. The joint use of these characteristics results in an intelligent encoding which exploits the problem-specific knowledge in an efficient way. We have also extended the genetic crossover and mutation operators in order to work up on those solutions, and recombine in a beneficial way the parents' information to form the offspring.

We have hybridized the algorithm incorporating in the genetic process two different procedures. First of all, a local search procedure, based on forward and backward improvement is applied to solutions in order to reduce the project makespan. This process is applied taking into account the additional genes of the new encoding. Secondly, a random replacement procedure permits us to randomly introduce variability into the population, in order to reintroduce lost genetic material and avoid being trapped in a local optimum.

An extensive computational experience has been carried out in order to test the performance of the new encoding, the new genetic operators and the procedures proposed. Standard sets of instances of PSPLIB have been used for this comparison, establishing as stopping criteria the construction of 1,000, 5,000 and 50,000 schedules. First of all, we have analyzed the performance of the encoding and the genetic operators, comparing the new algorithm with our previous genetic algorithm, the only differences between both algorithms being the encoding and the crossover and mutation operators. The results show that the use of the new representation and the extended operators lead to a better performance of the algorithm, irrespectively of the instances set solved or the number of schedules computed.

Next we have tested the performance of the two improvement processes. The local search procedure has demonstrated its improvement power when applied to all the generations in the process. There are no large differences between applying the procedure in one or two phases, although we fixed the application of one phase to all the generations in order to establish a configuration irrespective of the project characteristics. With respect to the random replacement procedure, the results show that it is efficient when solving small or medium-size projects, but not so for large projects. Large projects are very difficult to solve and need many more generations to converge, so this procedure is not advisable.

Once the algorithm has been calibrated and configured, we have compared its performance with some of the best and more recent algorithms, including other genetic algorithms, simulated annealing, tabu search, scatter search and sampling mechanisms. The computational results show that there are three algorithms which are better than the rest. These are the algorithm of Kochetov and Stoylar, the scatter search method of Debels et al and the genetic algorithm presented in this work.

Future research could include the use of the encoding designed in this work in other paradigms (simulated annealing, tabu search, ant systems,...) to solve the RCPSP or even to solve other scheduling problems. The algorithm proposed in this work could be adapted to solve the multi-mode version of the problem or problems which consider other objectives.

References

Alcaraz, J. (2001). Algoritmos Genéticos para Programación de Proyectos con Recursos Limitados, *Bell and Howell*, Valencia.

Alcaraz, J. and Maroto C. (2001). A robust genetic algorithm for resource allocation in project scheduling, *Annals of Operations Research* 102:83–109.

Alcaraz, J., Maroto C. and Ruiz R. (2003). Solving the multi-mode resource-constrained project scheduling problems with genetic algorithms, *Journal of the Operational Research Society* 54:614–626.

Artigues, C., Michelon, P. and Reusser, S. (2003). Insertion techniques for static and dynamic resource-constrained project scheduling, *European Journal of Production Research* 149:249–267.

Bouleimen, K. and Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple modes version, *European Journal of Operational Research* 149:268–281.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1998). Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* 112:3–41.

Debels, D., De Reyck, B., Leus, R. and Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta-heuristic for project scheduling, *European Journal of Operational Research* 169(2):638–653.

Elmaghraby, S.E. (1977). Activity Networks: Project Planning and Control by Network Models, *Wiley*, New York.

Fleszar, H. and Hindi, K. (2004). Solving the resource-constrained project scheduling problem by a variable neighbourhood search, *European Journal of Operational Research* 155:402–413.

Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning, *Addison Wesley*.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling, *Naval Research Logistics* 45:733–750.

Hartmann, S. (1999). Project Scheduling under Limited Resources, *Springer*.

Hartmann, S. (2002). A self adapting genetic algorithm for project scheduling under resource constraints, *Naval Research Logistics* 49:433–448.

Hartmann, S. and Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem-a survey of recent developments, *European Journal Operational Research* 127:394–407.

Herroelen, W., Demeulemeester, E. and De Reyck, B. (1998). A classification scheme for project scheduling, in: *Project Scheduling: Recent Models, Algorithms and Applications*, J. Weglarz, ed., Kluwer Academic Publishers, Berlin, pp. 1–26.

Hindi, K., Yang, H. and Fleszar, K. (2002). An evolutionary algorithm for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation* 6:402–413.

Jozefowska, J., Mika, M., Rozycki, R., Waligora, G. and Weglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling, *Annals of Operations Research* 102:137–155.

Kelley, J.E. (1963). The critical-path method: Resources planning and scheduling, in: *Industrial Scheduling*, Muth, J.F. and Thompson, G.L., eds., Prentice-Hall, New Jersey, pp. 347–365.

Kim, K.W., Yun, Y.S., Yoon, J.M., Gen, M. and Yamazaki, G. (2005). Hybrid genetic algorithm with adaptive abilities for resource-constrained multiple project scheduling, *Computers in Industry* 56:143–160.

Klein, R. (2000). Project scheduling with time-varying resource constraints, *International Journal of Production Reseach* 38:3937–3952.

Kochetov, Y. and Stoylar, A. (2003). Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem, in: *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*, Russia.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, *European Journal of Operational Research* 90:320–333.

Kolisch, R. and Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis, in: *Project Scheduling: Recent Models, Algorithms and Applications*, J. Weglarz, ed., Kluwer Academic Publishers, Berlin, pp. 147–178.

Kolisch, R. and Hartmann, S. (To appear), Experimental investigation of heuristics for resource-constrained project scheduling: an update, *European Journal of Operational Research.*

Kolisch, R. and Sprecher, A. (1996). PSPLIB - a project scheduling problem library, *European Journal of Operational Research* 96:205–216.

Nonobe, K. and Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, eds, Kluwer Academic Publishers, pp. 557–588.

Özdamar, L. (1999). A genetic algorithm approach to a general category project scheduling problem, *IEEE Transactions on Systems, Man, and Cybernetics* 29:44–59.

Sampson, S.E. and Weiss, E.N. (1993). Local search techniques for the generalized resource constrained project scheduling problem, *Naval Research Logistics* 40:665–675.

Sprecher, A., Kolisch, R. and Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem, *European Journal of Operational Research* 80:94–102.

Thomas, P.R. and Salhi, S. (1997). An investigation into the relationship of heuristic performance with network-resource characteristics, *Journal of the Operational Research Society* 48:34–43.

Toklu, Y.C. (2002). Application of genetic algorithms to construction scheduling with or without resource constraints, *Canadian Journal of Civil Engineering* 29:421–429.

Tormos, P. and Lova, A. (2001). A competitive heuristic solution technique for resource-constrained project scheduling, *Annals of Operations Research* 102:65–81.

Tormos, P. and Lova, A. (2003). An efficient multi-pass heuristic for project scheduling with constrained resources, *International Journal of Production Research* 41:1071–1086.

Valls, V., Ballestin, F. and Quintanilla, M.S. (2005). Justification and RCPSP: A technique that pays, *European Journal of Operational Research* 165:375–386.

Valls, V., Quintanilla, M.S. and Ballestin, F. (2003). Resource-constrained project scheduling: A critical reordering heuristic, *European Journal of Operational Research* 149:282–301.

Zhang, H., Li, X., Li, H. and Huang, F. (2005). Particle swarm optimization-based schemes for resource-constrained project scheduling, *Automation in Construction* 14:393–404.

Chapter 11

POPULATION LEARNING ALGORITHM FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING

Piotr Jedrzejowicz, Ewa Ratajczak
Department of Information Systems
Gdynia Maritime University, Poland
{pj, ewra}@am.gdynia.pl

Abstract The paper proposes applying the population-learning algorithm to solving both the single-mode and the multi-mode resource-constrained pro-ject scheduling problems (denoted as RCPSP and MRCPSP, respectively) with makespan minimization as an objective function. The paper contains problem formulation and a description of the proposed implementation of the population learning algorithm (PLA). To validate the approach a computational experiment has been carried out. It has involved 1440 instances of the RCPSP and 3842 instances of the MRCPSP obtained from the available benchmark data sets. Results of the experiment show that the proposed PLA implementation is an effective tool for solving the resource-constrained project scheduling problems. In case of the RCPSP instances the algorithm in a single run limited to 50000 solutions generated has produced results close to the results of the best known algorithms as compared with average deviation from critical path. In case of the MRCPSP instances the proposed algorithm in a single run has produced solutions with mean relative error value below 1.6% as compared with optimal or best known solutions for benchmark problems.

Keywords: Project scheduling, RCPSP, MRCPSP, Population Learning Algorithm.

11.1 Introduction

The paper proposes applying the population-learning algorithm (PLA) to solving instances of the single-mode and the multi-mode resource-con-strained project scheduling problems (denoted as RCPSP and MRCPSP, respectively) with makespan minimization as an objective function. In the single-mode case a project consists of a set of activities, where each activity has to be processed in a single, prescribed way (mode). Each activity requires some resources, availability of which is constrained. The discussed problem is computationally difficult and belongs to the NP-hard class. Because of its practical importance RCPSP has attracted a lot of attention and many exact and heuristic methods have been proposed for solving it (see for example Davis and Heidorn (1971), Hartmann (2001), Hartmann and Drexl (1998), Hartmann and Kolisch (2005)). In the multi-mode case activities can be executed in one out of several modes. The modes reflect alternative combinations of resource quantities employed to fulfill the activities. As it was observed in Sprecher and Drexl (1998), in such a case the activity duration is a discrete function of the employed quantities, that is, using this concept e.g. working-off an activity can be accelerated by raising the quantities coming into operation (time/resource trade-off). Moreover, by raising the quantities of some resources and reducing the quantities of others resource substitution (resource/resource trade-off) can be realized.

Exact algorithms seem suitable only for solving relatively small instances of the RCPSP and MRCPSP. For larger and more realistic instances an approach based on approximate algorithms is required. Such algorithms can be evaluated experimentally. Usual approach is to use the existing set of benchmark instances with known lower bounds, optimal solutions or upper bounds. Criteria for such an evaluation include, usually, two factors - quality of solutions obtained and computational effort required. Interesting evaluation of the heuristic and meta-heuristic algorithms for the resource-constrained project scheduling problem can be found in Hartmann and Kolisch (1999), Hartmann and Kolisch (2000) and Hartmann and Kolisch (2005). Some recent approaches based on hybrid algorithms and metaheuristics yield a very competitive solutions Jozefowska et al (2001), Nonobe and Ibaraki (2002), Debels et al (2004), Debels and Vanhoucke (2005), Valls et al (2004).

In this paper an approach based on implementation of the population-learning algorithm belonging to the class of population-based methods is proposed. The PLA has proven quite successful in solving some other difficult scheduling problems (for example see Jedrzejowicz and Jedrzejowicz (2002)). The paper is organized as follows: Section 11.2 includes problem formulation. Section 11.3 presents main features of the population-learning algorithm. Section 11.4 contains details of the proposed implementations of PLA designed to solving both - the single-mode and the multi-mode resource-constrained project scheduling problems. Section 11.5 presents validating experiment and its results. Section 11.6 includes conclusions and suggestions for future research.

11.2 Problem formulation

A project that belongs to the class of the single-mode resource-con-strained project scheduling problem consists of a set of n activities, where each activity has to be processed without interruption to complete the project. The dummy activities 1 and n represent the beginning and the end of the project. The duration of an activity j, $j = 1, \ldots, n$ is denoted by d_j where $d_1 = d_n = 0$. There are r renewable resource types. The availability of each resource type k in each time period is $\check{r}_k$ units, $k = 1, \ldots, r$. Each activity j requires r_{jk} units of resource k during each period of its duration where $r_{1k} = r_{nk} = 0$, $k = 1, \ldots, r$. All parameters are non-negative integers. There are precedence relations of the finish-start type with a zero parameter value (i.e. $FS = 0$,) defined between the activities. In other words activity i precedes activity j if j cannot start until i has been completed. The structure of a project can be represented by an activity-on-node network $G = (SV, SA)$, where SV is the set of activities and SA is the set of precedence relationships. $SS_j(SP_j)$ is the set of successors (predecessors) of activity j, $j = 1, \ldots, n$. It is further assumed that $1 \in SP_j$, $j = 2, \ldots, n$, and $n \in SS_j$, $j = 1, \ldots, n-1$. The objective is to find a schedule S_s of activities starting times $[s_1, \ldots, s_n]$, where $s_1 = 0$ and resource constraints are satisfied, such that the schedule duration $T(S_s) = s_n$ is minimized.

It has been shown in Blazewicz et al (1983) that the above formulated RCPSP as a generalization of the classical job shop scheduling problem, belongs to the class of NP-hard optimization problems. Therefore, heuristic solution procedures are indispensable when solving large problem instances as they usually appear in practical cases.

In case of the multi-mode resource-constrained project scheduling problem each activity j, $j = 1, \ldots, n$ may be executed in one out of M_j modes. The activities may not be preempted and a mode once selected may not change, i.e., a job j once started in mode m has to be completed in mode m without interruption. Performing job j in mode m takes d_{jm} periods and is supported by a set R of renewable, a set N of non-renewable and a set D of doubly constrained resources. Considering a time horizon, that is, an upper bound T on the project's makespan, one has an available amount of renewable (doubly constrained) resource as well as certain overall capacity of the non-renewable (doubly constrained) resource. Clearly, since the doubly constrained resources can easily be taken into account by appropriately enlarging the sets of renewable and non-renewable resources they do not have to be considered explicitly. The parameters are assumed as integer-valued. The objective is to find a makespan minimal schedule that meets the constraints imposed by the precedence relations and the limited resource availabilities. It is obvious that the multi-mode problem can not be computationally easier than the RCPSP.

11.3 Population Learning Algorithm

Population learning algorithm introduced originally in Jedrzejowicz (1999) is a popula-tion-based method inspired by analogies to a phenomenon of social education processes in which a diminishing number of individuals enter more and more advanced learning stages. PLA take advantage of the following features common to organized education systems:

- A huge number of individuals enter the system.
- Individuals learn through organized tuition, interaction, self-study, trials and errors.
- Learning process is inherently parallel (different schools, curricula, teachers, etc.).
- Learning process is divided into stages.
- More advanced stages are entered by a diminishing number of individuals from the initial population.
- At higher stages more advanced learning and improvement techniques are used.
- A final stage is reached by only a fraction of the initial population.

In PLA an individual represents a coded solution or part of it of the considered problem. Initially, a number of individuals, known as the initial population, is randomly generated or constructed using some construction heuristics. Once the initial population has been generated, individuals enter the first learning stage. It involves applying some, possibly basic and elementary, improvement schemes. These can be based, for example, on some local search procedures. The improved individuals are then evaluated and better ones pass to subsequent stages. A strategy of selecting better or more promising individuals at each stage must be defined and duly applied. At following stages the whole cycle is repeated. Individuals are subject to improvement and learning, either individually or through information exchange, and the selected ones are again promoted to a higher stage with the remaining ones dropped-out from the process. At the final stage the remaining individuals are reviewed and the best one represents a solution to the problem at hand.

At different stages of the process, different improvement schemes and learning procedures are applied. These gradually become more and more sophisticated and time consuming as there are less and less individuals to be taught. Basic PLA design elements are presented in Table 11.1. General idea of the PLA approach is shown in Figure 11.1. Several successful PLA applications were described in Czarnowski et al (2001)b, Czarnowski et al (2001), Czarnowski et al (2001a), Jedrzejowicz (1999), Jedrzejowicz and Skakowski (2000).

Table 11.1. Population learning algorithm design elements.

Design element	Comment
Definiton of an individual	An individual is a coded feasible solution or part of a solution. The designer should aim at achieving ease of manipulation and storage, ease of transformation into a solution and ease of fitness evaluation.
Procedure for generating the initial population of individuals	The designer should aim at achieving unbiased individuals, if possible representing all regions of the feasible solution space. A good construction heuristics can be used to produce a seed from which the initial population can be generated.
Size of the initial population	Must be set at the PLA fine-tuning stage. Should be chosen as a compromise between the requirement of the sufficient and adequate representation assuring good quality of results and the available computational resources.
Number of learning and improvement stages	Depends on availability of learning and improvement algorithms. Should be chosen with a view of finding a satisfactory compromise between quality of solutions and computation time.
Number and size of parallel groups of individuals at each stage	Should be chosen considering the available computational resources, complexity of the problem at hand and complexity of the learning and improvement algorithms used. Another compromise between quality of solutions and computation time is involved.
Learning and improvement algorithms for each stage/each group	Basic and simple procedures at earlier stages should be replaced by more complex and sophisticated at later ones. The designer should try to assure adequate diversity of learning and improvement algorithms used.
Fitness function of an individual	Should be simple (easily computable) and directly related to the quality of the solution represented by an individual.
Selection strategy	Must be set at the PLA fine-tuning stage. Rules for rejection and promotion of individuals may differ at various stages. The designer should aim at achieving good efficiency of the algorithm not loosing, too early, representation sufficiency and adequacy. Another compromise between quality of solutions and computation time is involved.

Figure 11.1 covers a simple, non-parallel version of the algorithm. For a parallel PLA the following features need to be defined:

- A set of rules controlling how individuals are grouped into concurrent populations at various stages.
- The respective rules for running learning and improvement algorithms in parallel at various stages.

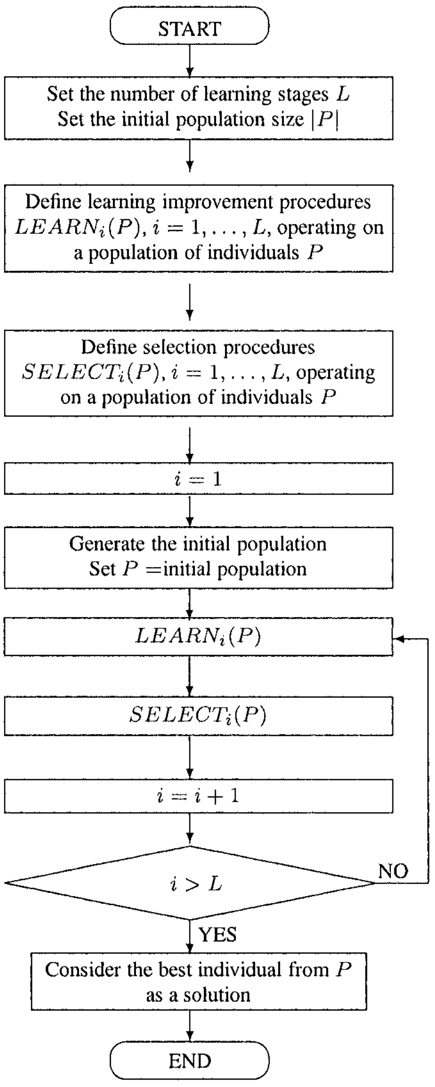

START
Set the number of learning stages L
Set the initial population size $|P|$
Define learning improvement procedures $LEARN_i(P), i = 1, \dots, L$, operating on a population of individuals P
Define selection procedures $SELECT_i(P), i = 1, \dots, L$, operating on a population of individuals P
$i = 1$
Generate the initial population
Set P =initial population
$LEARN_i(P)$
$SELECT_i(P)$
$i = i + 1$
$i > L$
NO
YES
Consider the best individual from P as a solution
END

Figure 11.1. General idea of the population learning algorithm.

- Rules for information exchange and coordination between concurrent processes.

Designing population learning algorithm intended for solving a particular problem type allows the designer a lot of freedom (as, in fact, happens in case of majority of other population-based algorithms). Moreover, an effective PLA would certainly require a lot of fine-tuning and experimenting. This could be considered as a disadvantage, at least as long as the process of setting different parameters of the population learning algorithm is rather based on heuristics instead of some theoretical or statistical rules, which, unfortunately, are not yet appropriately developed. Main PLA design elements are summarized in Table 11.1.

PLA shares some features with evolutionary programs as discussed, for example, in Michalewicz (1992). The idea of refining population of solutions during the subsequent computation stages is common to PLA and memetic algorithms Moscato (1999). The latter, however, assume a constant population size and a single local search procedure and relay to a greater extend on typical genetic/evolutionary algorithms operators. There are also some similarities between PLA and cultural algorithms where an inheritance process operates at the micro-evolutionary levels Reynolds (1994).

11.4 PLA implementations

11.4.1 PLA for RCPSP

The proposed implementation involves three learning and improvement stages. Value of the goal function is directly used as a measure of quality of individuals and hence as a selection criterion. An individual in the algorithm is a schedule represented as a vector of activities $S = [a_1 \dots, a_n]$, each activity a_j being an object consisting of: starting time - s_j, duration - d_j , set of required units of resources, set of predecessors - SP_j and set of successors - SS_j. An individual is represented as an ordered activity list, in which for each activity, all its predecessors are placed at earlier positions and all its successors at later positions on the list. The list serves as a starting point for generating a solution using heuristic known as the serial SGS (Schedule Generation Scheme) described in Hartmann and Kolisch (1999).

The algorithm requires that values of the following parameters are set:

- p - multiplier used to calculate the size of an initial population.
- $xi1$ - coefficient used to calculate the number of iterations at the first learning and improvement stage.
- $xi2$ - coefficient used to calculate the number of iterations at the second learning and improvement stage.

Values of the above control parameters are set at the algorithm fine-tuning phase. All random moves within the algorithm are drawn from the uniform distribution. All parameter values used in the following pseudo-code have been chosen by trials and errors during the fine-tuning phase. The pseudo-code of the proposed PLA algorithm is shown in Figure 11.2. In the pseudo-code P denotes population, and $|P|$ size of the population P.

```
PLA:
  Set size of $P$, $|P| = p \cdot n$;
  Set $xi1$, $xi2$;
  Create initial population $P$;
  /* the first learning stage */
  for $it := 1$ to $xi1 \cdot n$ do
    for $i := 1$ to $40\% \cdot |P|$ do
      Crossover(Random($P$), Random($P$));
    for $i := 1$ to $5\% \cdot |P|$ do
      RHIA(Random($P$),2,0.25 $\cdot |P|$);
    for $i := 1$ to $5\% \cdot |P|$ do
      LSA(Random($P$),2,6,10);
  Selection(medium_makespan);
  /* the second learning stage */
  for $it := 1$ to $xi2 \cdot n$ do
    for $i := 1$ to $40\% \cdot |P|$ do
      Crossover(Random($P$), Random($P$));
    for $i := 1$ to $5\% \cdot |P|$ do
      Mutation(Random($P$));
    for 2 best solutions $S \in P$ do
      EPTA($S$,5,1);
    for $i := 1$ to $2\% \cdot |P|$ do
      LSA(Random($P$),2,6,10);
  Selection(medium_makespan);
  /* the third learning stage */
  for each solution $S \in P$ do
    EPTA($S$,6,2);
    LSA($S$,10,2,10);
End pseudo-code.
```

Figure 11.2. Pseudocode of the PLA procedure.

The algorithm creates an initial population by producing four individuals using simple construction heuristics and generating randomly the remaining ones. Heuristics are based on the following rules:

- Shortest duration first.
- Shortest duration last.
- Longest duration first.
- Longest duration last.

The first learning stage uses evolutionary operators and a simple local search algorithm (LSA). Three procedures: Crossover, RHIA and LSA are repeated

$xil \cdot n$ times. The Random(P) function chooses randomly an individual from the population. A pseudocode of the above procedures are shown in Figure 11.3.

```
***Crossover***($S_1$,$S_2$):
  Do simple (one point) crossover on two randomly chosen
  individuals $S_1, S_2 \in P$;
End pseudo-code.

***RHIA***($S$,$itNumber$,$iRange$):
  **for** $it := 1$ **to** $itNumber$ **do**
    Choose randomly $iRange$ homogenous intervals in solution $S$
    and try to improve the resource utilisation by applying
    Resource_Utilisation_Improving function Valls et al (2004);
End pseudo-code.

***LSA***($S$,$itNumber$,$iStep$,$fStep$):
  **for** $it := 1$ **to** $itNumber$ **do**
    **for** $s := iStep$ **to** $fStep$ **do**
      **for** $j := 1$ **to** scheduleLength($S$) $- s$ **do**
        Exchange activity from position $j$ with the activity
        from position $j + s$ in $S$;
        **if not** the new solution is better
          **then** recall the exchange;
End pseudo-code.
```

Figure 11.3. Pseudocode of the Crossover, Mutation and LSA procedures.

The RHIA procedure is based on the idea of improving the resource utilisation in a homogeneous intervals proposed in Valls et al (2004). The proposed function Resource_Utilisation_Improving is used in this part of PLA as a kind of mutation operator. $itNumber$ denotes the number of iteration for the procedure and $iRange$ is the number of improving intervals.

The LSA procedure requires four variables. The first, S denotes an individual, second ($itNumber$) is the number of iteration for the procedure. The last two indicate the initial and final distance between activities under exchange.

The second learning stage uses Crossover, Mutation and two heuristics - EPTA (exact precedence tree algorithm) and LSA. A pseudocode of the Mutation and EPTA procedures are prsented in Figure 11.4. EPTA is based on the precedence tree approach proposed in Sprecher and Drexl (1998). It finds an optimum solution by enumeration for a partition of the schedule consisting of some activities, which number is denoted as $partExtent$. In the following pseudocode variable S denotes an individual, $|S| = n$. The $step$ variable denotes the distance between starting points of the considered partitions.

Finally, the third learning stage uses two heuristics EPTA and LSA, both with different values of parameters as compared with settings in the earlier stages, which results in more iterations and higher granularity of the neighbourhood explored.

```
Mutation($S$):
    Move an activity from randomly chosen position in the
    schedule $S$ to another randomly chosen position in this
    schedule, in such a way that no successor of this activity
    can be found before it;
End pseudo-code.

EPTA($S$,$partExtent$,$step$):
    $i := 1$;
    while $i \cdot step + partExtent < n$ do
        Find an optimal solution for a part of the schedule
        beginning from activity on position $i \cdot step$ and ending
        in activity on position $i \cdot step + partExtent$;
End pseudo-code.
```

Figure 11.4. Pseudocode of the EPTA procedure.

11.4.2 PLA for MRCPSP

In case of the MRCPSP an individual is represented as in RCPSP case but additionally for each activity on the list its mode set is remembered and the mode choosen to execute is considered in SGS.

The overall scheme of the PLA implementation for solving the MRCPSP instances is an extension of the PLA implementation used in the RCPSP case. PLA-MRCPSP uses identical number of stages and similar learning/improvement procedures. Differences with respect to the PLA-RCPSP are summarized below.

In case of the MRCPSP a preprocessing phase is carried out. It involves removing all non-executable modes from the project data, deleting the redundant non-renewable resources and eliminating all inefficient modes (for details see Hartmann and Drexl (1998)). Additionally, for each data set the longest chain of predecessors-successors is serched and remembered. The chain represents the longest list of subsequent activities, which are tied together through a set of precedence relationships. If there are more then one such chains, a new one combining all activities common to original chains is produced. Within the discussed implementation also another mutation operator has been integrated. It operates through randomly changing an activity mode. There are also some changes within the local search procedure used at each of the stages. Local search procedure for the multi-mode version is shown in Figure 11.5.

```
LSA-MRCPSP($S$, $itNumber$, $iStep$, $fStep$):
  for $it := 1$ to $itNumber$ do
    for $s := iStep$ to $fStep$ do
      for $j := 1$ to scheduleLength($S$) $- s$ do
        for $k_j := 1$ to number of modes for activity $j$ do
          for $k_{j+s} := 1$ to number of modes for activity $j + s$ do
            Exchange activity from position $j$ with the
            activity from position $j + s$ with eventually
            exchnging at the same time mode of the activity
            at position $j$ for $k_j$ and at position $j + s$ for $k_{j+s}$;
            if not the new solution is better
            then recall the exchange;
End pseudo-code.
```

Figure 11.5. Pseudocode of the LSA-MRCPSP procedure.

Furthermore, while executing the PLA-MRCPSP, each individual emerging modified as a result of some random move, that is either mutation or crossover, or as a result of applying a heuristic, that is either LSA or EPTA, is checked with respect to its feasibility in terms of non-renewable resources requirement. After this a three stage improvement process is carried. It is aiming at:

1. Decreasing the use of non-renewable resources.
2. Decreasing an excessive demand for non-renewable resources.
3. Decreasing an activity execution time through changing its mode without violating constraint on non-renewable resources.

Stage 1 is based on a simple heuristic. Considering in turn all activities from the activity list, the activity, for which changing a mode to another one produces a maximum profit, is selected and the respective change is accepted. The notion of profit is understood here as a difference between use of non-renewable resources as required in both compared modes.

Stage 2 is carried out only if executing stage 1 has not produced a feasible solution in terms of requirements for non-renewable resources. It aims at selecting an activity for which changing its mode produces the maximum decrease of excess in demand for non-renewable resources. Search for such an activity/node is being repeated until there is no more excessive demand for non-renewable resources or all n activities on the list have been already considered.

Stage 3 is carried out only if executing the earlier stages has produced a feasible solution in terms of requirements for renewable resources. It aiming at decreasing the execution time of an activity is based on two sub procedures:

- From all activities on the list the one for which changing its mode brings about a maximum decrease of the execution time without violating constraint on non-renewable resources is selected. The search is repeatedly carried out until an activity and the respective modes, leading to a maximum decrease of the processing time of all activities, is found.
- The sub procedure described above is executed for activities from the longest chain of predecessors - successors.
- If the attempt at the stage 3 is not successful a solution obtained in stage 2 remains uchanged.

Both sub procedures are executed and the best result in terms of time gains out of the all results produced is accepted and the respective individual is added to the population of individuals.

11.5 Validating experiment

11.5.1 The RCPSP case

To validate the proposed approach computational experiment has been carried out using 1440 benchmark instances of single-mode RCPSP. The benchmark data set used in the reported experiments includes 480 instances for each out of the three problem sizes (30, 60, 90 and 120 activities). The benchmark data set together with known optimal solution values/upper bounds can be found at http://129.187.106.231/psplib.

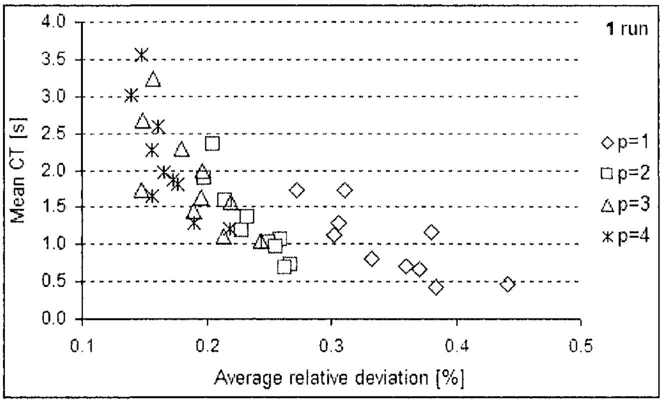


Figure 11.6. Setting value of the multiplier p (single run)

The fine-tuning phase of the experiment has been devoted to finding by trials and errors values of the PLA parameters assuring acceptable compromise between computation time and quality of solutions. This search has been carried out using the subset of available benchmark instances consisting of all instances of the RCPSP with 30 activities. Figures 11.6 and 11.7 show example results

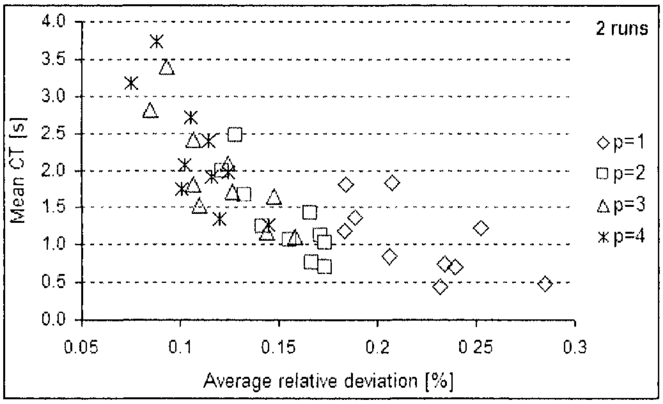


Figure 11.7. Setting value of the multiplier p (two runs)

with respect to setting value of the multiplier p. The final settings include selected combination of parameter values considered in the experiment $xi1 = xi2 = \frac{2}{3}$ and multiplier $p = 3$ used to calculate the population size ($|P| = p \cdot n$).

Table 11.2. Experiment results, RCPSP (30 activities), single PLA run, relative deviation from the optimal solution.

Max number of schedules	*single PLA run*				
	Avg. rel. dev.	*Max rel. dev*	*Equal opt.*	*Mean CT[s]*	*Max CT[s]*
1000	0.60%	9.52%	79.17%	0.20	0.56
5000	0.19%	4.26%	90.42%	1.26	15.78
50000	0.12%	3.45%	92.92%	2.67	12.40

The experiment involved solving all 1440 benchmark instances twice in two runs. The results are evaluated in terms of average and maximum relative deviation from the optimal makespan or, if unknown, from the critical path lower bound (Avg. rel. dev, Max rel. dev.), percent of solutions equal to respective optimal solutions (Equal opt.) as well as mean and maximum computation time (Mean CT, Max CT) for a single instance. Experiment results for the three stopping criteria (limited the number of schedules to 1000, 5000 and 50000, respectively) are shown in Tables 11.2 - 11.6.

Experiment has been carried on the PC computer with the AMD XP 2600+ processor.

Table 11.7 shows average deviation from the best solutions produced by the best metaheuristics reported in Hartmann and Kolisch (2005).

11.5.2 The MRCPSP case

The experiment involving multi-mode instances has been carried on PC with the AMD XP 2600+ processor. Benchmark data set has been obtained from the PSPLIB (http://129.187.106.231/psplib/). About 550 instances for each out of 7 classes of the activities sizes have been solved. Altogether 3842 instances have been solved for each of the investigated combinations of the PLA parameters. In addition to the earlier investigated parameter combinations additional one with $p = 6$ and set "b" has been added. The respective results are shown in Tables 11.8 - 11.14.

Single and parallel run experiment results involving all 3842 MRCPSP instances prove that the PLA produces good results in a remarkably short time. Quality of the results obtained is, however, not fully satisfactory. To remedy this situation it has been decided to test a parallel PLA implementation in which

Table 11.3. Experiment results, RCPSP (30 activities), two PLA runs, relative deviation from the optimal solution.

Max number of schedules	*two PLA runs*				
	Avg. rel. dev.	*Max rel. dev*	*Equal opt.*	*Mean CT[s]*	*Max CT[s]*
1000	0.45%	6.90%	81.46%	0.21	0.59
5000	0.13%	3.45%	92.71%	1.32	6.07
50000	0.08%	3.45%	95.00%	2.81	13.02

Table 11.4. Experiment results, RCPSP (60 activities), relative deviation from the critical path lower bound.

Max number of schedules	*single PLA run*			*two PLA runs*		
	Avg. rel. dev.	*Mean CT[s]*	*Max CT[s]*	*Avg. rel. dev.*	*Mean CT[s]*	*Max CT[s]*
1000	13.33%	0.26	0.74	12.50%	0.27	0.78
5000	12.86%	0.84	3.41	12.07%	0.88	3.58
50000	11.52%	14.86	74.84	10.77%	15.61	78.59

Table 11.5. Experiment results, RCPSP (90 activities), relative deviation from the critical path lower bound.

Max	*single PLA run*			*two PLA runs*		
number of schedules	*Avg. rel. dev.*	*Mean CT[s]*	*Max CT[s]*	*Avg. rel. dev.*	*Mean CT[s]*	*Max CT[s]*
1000	13.32%	0.50	1.53	12.39%	0.53	1.61
5000	13.07%	1.09	4.42	12.20%	1.15	4.64
50000	11.92%	7.98	31.53	11.16%	8.37	33.11

Table 11.6. Experiment results, RCPSP (120 activities), relative deviation from the critical path lower bound.

Max	*single PLA run*			*two PLA runs*		
number of schedules	*Avg. rel. dev.*	*Mean CT[s]*	*Max CT[s]*	*Avg. rel. dev.*	*Mean CT[s]*	*Max CT[s]*
1000	35.83%	1.87	2.69	35.11%	1.97	2.82
5000	35.12%	3.42	5.89	34.56%	3.59	6.19
50000	33.88%	23.71	53.92	32.89%	24.90	56.61

Table 11.7. Average relative deviation from the critical path lower bound - the best results presented in Hartmann and Kolisch (2005) as compared with the PLA performance.

Algorithm	*Number of activities*	*Max number of schedules* 1000	5000	50000
GA, TS-path relinking	30	0.10%	0.04%	0.00%
	60	11.71%	11.17%	10.74%
	120	34.74%	33.36%	32.06%
GA-hybrid, FBI	30	0.27%	0.06%	0.02%
	60	11.56%	11.10%	10.73%
	120	34.07%	32.54%	31.24%
GA-forw.-backward	30	0.33%	0.12%	–
	60	12.57%	11.86%	–
	120	39.36%	36.57%	–
PLA	30	0.60%	0.19%	0.12%
	60	13.33%	12.86%	11.52%
	120	35.83%	35.12%	33.88%

Table 11.8. Experiment results, MRCPSP (10 activities), relative deviation from the optimal solution.

Max	*single PLA run*				*two PLA runs*			
number of schedules	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*
1000	3.44%	68.56%	0.13	1.44	2.89%	72.39%	0.13	1.51
5000	0.71%	88.80%	0.33	2.24	0.39%	92.91%	0.34	3.50
50000	0.67%	88.81%	0.33	2.92	0.36%	92.91%	0.35	3.50

Table 11.9. Experiment results, MRCPSP (12 activities), relative deviation from the optimal solution.

Max	*single PLA run*				*two PLA runs*			
number of schedules	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*
1000	3.65%	59.23%	0.15	0.69	2.99%	64.17%	0.16	0.73
5000	0.92%	84.83%	0.52	2.10	0.54%	90.31%	0.55	2.20
50000	0.87%	85.10%	0.58	2.10	0.50%	90.49%	0.58	2.20

Table 11.10. Experiment results, MRCPSP (14 activities), relative deviation from the optimal solution.

Max	*single PLA run*				*two PLA runs*			
number of schedules	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*
1000	4.06%	53.09%	0.18	0.52	3.53%	56.26%	0.19	0.54
5000	1.24%	77.68%	0.66	3.65	0.81%	84.21%	0.69	3.83
50000	0.95%	81.40%	0.91	6.83	0.62%	86.93%	0.95	7.17

Table 11.11. Experiment results, MRCPSP (16 activities), relative deviation from the optimal solution.

Max	*single PLA run*				*two PLA runs*			
number of schedules	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*
1000	4.39%	50.18%	0.20	0.79	3.77%	54.36%	0.21	0.83
5000	1.51%	71.82%	0.75	4.03	1.13%	77.09%	0.79	4.23
50000	1.02%	78.45%	1.34	5.31	0.75%	82.55%	1.41	5.58

Table 11.12. Experiment results, MRCPSP (18 activities), relative deviation from the optimal solution.

Max	*single PLA run*				*two PLA runs*			
number of schedules	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*
1000	4.51%	48.93%	0.19	0.44	4.56%	50.18%	0.20	0.46
5000	1.63%	66.82%	0.71	2.46	1.77%	67.51%	0.75	2.58
50000	1.06%	76.99%	2.71	27.33	0.75%	82.31%	2.85	28.69

Table 11.13. Experiment results, MRCPSP (20 activities), relative deviation from the optimal solution.

Max	*single PLA run*				*two PLA runs*			
number of schedules	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*	*Avg. rel. dev.*	*Equal opt.*	*Mean CT [s]*	*Max CT [s]*
1000	4.74%	51.54%	0.19	0.59	4.04%	55.07%	0.20	0.62
5000	1.88%	68.30%	0.73	3.12	1.42%	77.73%	0.77	3.28
50000	1.09%	76.45%	1.84	9.36	0.75%	82.43%	1.93	9.83

Table 11.14. Experiment results, MRCPSP (30 activities), relative deviation from the best known solution.

Max	*single PLA run*			*two PLA runs*		
number of schedules	*Avg. rel. dev.*	*Mean CT[s]*	*Max CT[s]*	*Avg. rel. dev.*	*Mean CT[s]*	*Max CT[s]*
1000	6.07%	0.23	0.66	5.50%	0.24	0.69
5000	4.41%	0.64	2.55	3.87%	0.67	2.67
50000	1.57%	5.67	22.64	1.31%	5.96	23.77

Table 11.15. Average relative deviation from the optimal or best known solution for Simulated Annealing algorithm proposed in Jozefowska et al (2001) as compared with the PLA performance.

Number	*SA algorithm Jozefowska et al (2001)*			*PLA*		
of	*Max number of schedules*			*Max number of schedules*		
activities	1000	5000	50000	1000	5000	50000
10	5.17%	1.16%	0.23%	3.44%	0.71%	0.67%
12	7.55%	1.73%	0.37%	3.65%	0.92%	0.87%
14	10.75%	2.60%	0.24%	4.06%	1.24%	0.95%
16	12.05%	4.07%	1.06%	4.39%	1.51%	1.02%
18	12.63%	5.52%	0.56%	4.51%	1.63%	1.06%
20	16.00%	6.74%	0.80%	4.74%	1.88%	1.09%
30	17.10%	11.76%	3.81%	6.07%	4.41%	1.57%

each of the investigated PLA parameter variants is run as an independent agent on a PC within a cluster of computers. All such variants are run in parallel and the overall best solution is accepted. Computation time is equal to the computation time of the longest run variant plus a surcharge for finding overall best solution. The respective results are shown in Table 11.15. In Table 11.15 the state of the art results obtained in Jozefowska et al (2001) are compared with the PLA performance.

11.6 Conclusions

Experiment results show that the proposed PLA implementation is an effective tool for solving both single and multi-mode resource-constrained project scheduling problems. In case of the RCPSP instances the algorithm in a single run limited to 50000 solutions generated has produced results close to the results of the best known algorithms.

PLA results obtained in two runs are considerably better then single run ones. This proves that there is a room for further improvement of the PLA efficiency. Obvious solution would be a parallel PLA scheme. This has been attempted in case of the MRCPSP instances. Independently run PLA versions produced satisfactory results in a remarkably short time. Future research will concentrate on developing a parallel PLA with possible information exchange between parallel processes of learning and improvement. Another worthwhile direction of research should be concentrated on searching for effective and theoretically-based procedures for setting values of the PLA parameters. The most critical one seems to be the initial population size. It should not, however, be considered in isolation to other important variables like selection criteria, number of iterations at learning and improvement stages and granularity of local search procedures employed. Identifying relations between these factors as well as their joint influence on the quality of results should be the subject to further studies.

General conclusion of the paper can be formulated as follows: validating experiment results allow to consider the PLA as a useful and promising framework to deal with solving computationally difficult combinatorial problems with RCPSP and MRCPSP instances among them.

References

Blazewicz, J., Lenstra, J. and Rinnooy Kan, A. (1983). Scheduling subject to resource constraints: Classification and complexity, *Discrete Applied Mathematics* 5:11–24.

Czarnowski, I., Forkiewicz, M., Jedrzejowicz, P., Ratajczak, E., Skakowski, A. and Wierzbowska, I. (2001). Population-based scheduling on multiple processors, in: *Proc. 4th Metaheuristics Internetional Conference, MIC 2001*, Porto, Portugal, pp. 613–618.

Czarnowski, I., Gutjahr, W.J, Jedrzejowicz, P., Ratajczak, E., Skakowski, A. and Wierzbowska, I. (2001). Scheduling multiprocessor tasks in presence of the correlated failures, in: *The Proceedings of the Second International Workshop on Soft Computing Applied to Software Engineering*, SCASE, Enschede, The Netherlands.

Czarnowski, I., Jedrzejowicz, P. and Ratajczak, E. (2001). Population Learning Algorithm - example implementations and experiments, in: *Proc. 4th Metaheuristics Internetional Conference, MIC 2001*, Porto, Portugal, pp. 607–612.

Davis, E.W. and Heidorn, G.E. (1971). An algorithm for optimal project scheduling under multiple resource constraints, *Management Science*, 17:B803–B816.

Debels, D., De Reyck, B., Leus, R. and Vanhoucke, M. (2004). *A Hybrid Scatter Search / Electromagnetism Meta-Heuristic for Project Scheduling*, Univeriteit Gent, Faculteit Ekonomie en Bedrijfskunde Working Paper 04/237.

Debels, D. and Vanhoucke, M. (2005). A Bi-Population Based Genetic Algorithm for the Resource-Constrained Project Scheduling Problem, Univeriteit Gent, Faculteit Ekonomie en Bedrijfskunde Working Paper 05/294.

Hartmann, S. (2001). Project Scheduling with Multiple Modes: A Genetic Algorithm, *Annals of Operations Research* 102:11–135.

Hartmann, S. and Drexl, A. (1998). Project Scheduling with Multiple Modes: A Comparison of Exact Algorithms, *Networks* 32:283–297.

Hartmann, S. and Kolisch, R. (1999). Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, in: *Project scheduling: Recent models, algorithms and applications*, J. Weglarz ,ed, Kluwer, Amsterdam, pp. 147–178.

Hartmann, S. and Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, *European Journal of Operational Research* 127:394–407.

Hartmann, S. and Kolisch, R. (2005). Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update, *European Journal of Operational Research*, to apopear.

Jedrzejowicz, P. (1999). Social Learning Algorithm as a Tool for Solving Some Difficult Scheduling Problems, *Foundation of Computing and Decision Sciences*, 24(2):51–66.

Jedrzejowicz, J. and Jedrzejowicz, P. (2002). Permutation Scheduling Using Population Learning Algorithm, in: *Knowledge-Based Itelligent Information Engineering Systems and Allied Technologies*, E.Damiani et al, eds, IOS Press, Amsterdam, pp. 93–97.

Jedrzejowicz, P. and Skakowski, A. (2000). An island-based evolutionary algorithm for scheduling multiple-variant tasks, *Proceedings of ICSC Symposium Engineering of Intelligent Systems – EEIS'2000*, ICSC Academic Press, Paisley, pp. 1–9.

Jozefowska, J., Mika, M., Rozycki, R., Waligora, G. and Weglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling, *Annals of Operations Research*, 102:137–155.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin.

Moscato, P. (1999). Memetic Algorithms: A short introduction, in: *New Ideas in Optimization*, D.Corne, M.Dorigo, F.Glover, eds., McGraw-Hill, New York, pp. 219–234.

Nonobe, K. and Ibaraki T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, eds., Kluwer Academic Publishers, pp. 557–588.

Reynolds R.G. (1994). An Introduction to Cultural Algorithms, in: *Proceedings of the Third Annual Conference on Evolutionary Programming*, A.V. Sebald, L.J. Fogel, eds., World Scientific, River Edge, pp. 131–139.

Sprecher, A. and Drexl, A. (1998). Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm, *European Journal of Operational Research* 107:431–450.

Valls, V., Ballestin and F., Quintanilla, S. (2004). A Population-Based Approach to the Resource-Constrained Project Scheduling Problem, *Annals of Operations Research* 131:305–324.

Chapter 12

RESOURCE CONSTRAINED PROJECT SCHEDULING: A HYBRID NEURAL APPROACH

Selcuk Colak, Anurag Agarwal, Selcuk S. Erenguc
Department of Decision and Information Sciences, Warrington College of Business Administration, University of Florida, Gainesville, FL 32611, USA
scolak@ufl.edu, anurag.agarwal@cba.ufl.edu, selcuk.erenguc@cba.ufl.edu

Abstract This study proposes, develops and tests a hybrid neural approach (HNA) for the resource constrained project scheduling problem. The approach is a hybrid of the adaptive-learning approach (ALA) for serial schedule generation and the augmented neural network (AugNN) approach for parallel schedule generation. Both these approaches are based on the principles of neural networks and are very different from Hopfield networks. In the ALA approach, weighted processing times are used instead of the original processing times and a learning approach is used to adjust weights. In the AugNN approach, traditional neural networks are augmented in a manner that allows embedding of domain and problem-specific knowledge. The network architecture is problem specific and a set of complex neural functions are used to (i) capture the constraints of the problem and (ii) apply a priority rule-based heuristic. We further show how forward-backward improvement can be integrated within the HNA framework to improve results. We empirically test our approach on benchmark problems of size J30, J60 and J120 from PSPLIB. Our results are extremely competitive with existing techniques such as genetic algorithms, simulated annealing, tabu search and sampling.

Keywords: Project Management, Resource Constrained Project Scheduling, Neural Networks, Heuristics

12.1 Introduction

The resource-constrained project scheduling problem (RCPSP) is a well-known NP-Hard scheduling problem (Blazewicz et al (1983)). It is a classical problem in operations research with broad applicability in project management and production scheduling. It involves minimizing the makespan of a project by scheduling its activities which are subject to precedence and resource constraints. The amounts of available resources are fixed and known in advance. Resource requirements and processing times for each activity are deterministic and also known in advance and preemption of activities is not allowed. This problem has received the attention of many researchers for well over four decades. One of the recent research focuses in this area has been towards developing new metaheuristic approaches using artificial intelligence and/or biologically-inspired techniques. For solving this problem, two schedule generation schemes are commonly used – serial and parallel. In this work, we propose, develop and test a new hybrid metaheuristic approach based on the principles of neural networks. We use adaptive-learning approach (ALA) for serial and augmented-neural-network approach (AugNN) for parallel schedule generation scheme. We call our approach the hybrid-neural approach (HNA).

In the adaptive-learning approach (Agarwal et al (2005)), weighted processing times are used instead of the given processing times. Well-known heuristics are applied using these weighted processing times. An intelligent perturbation strategy used to adjust the weights allows non-deterministic local search. The AugNN approach was first applied to parallel schedule generation in the task-scheduling problem by Agarwal et al (2003). With suitable modifications, the AugNN approach can be applied to the parallel generation scheme for the RCPSP. The AugNN approach is quite different from the Hopfield network approach which has been applied to the traveling-salesman problem (Hopfield and Tank (1985) and job-shop scheduling (Sabuncuoglu and Gurgun (1996), Foo and Takefuji (1988)). In the AugNN approach, the traditional neural network is augmented to allow embedding of domain and problem-specific knowledge. The network architecture is designed to be problem specific; instead of the standard 3-layered network, it is a p-layered network, where p depends on the problem structure. Details will be explained in Section 16.4. Further, in the AugNN approach, the input, activation and output functions are complex functions, designed to (i) enforce the problem constraints, and (ii) apply a known priority heuristic. The AugNN approach, thus, allows incorporation of domain and problem-specific knowledge and affords the advantages of both the heuristic and iterative approaches. In this study, forward-backward improvement steps (Tormos and Lova (2001), Valls et al (2005) are also integrated within this framework of hybrid-neural approach.

We implement and test our proposed hybrid-neural approach on some well-known RCPS benchmark problem instances in the literature. Our results are very competitive with those of other techniques. Given that this approach is relatively new, it seems to hold a lot of promise; perhaps in future studies, it can be used in conjunction with other successful techniques, such as genetic algorithms, scatter search etc. to give improved results.

The rest of the paper is organized as follows. Section 16.2 presents a literature review for the RCPSP. We discuss how ALA is applied to the serial schedule generation problem in Section 16.3. The details of AugNN formulation for solving the parallel schedule generation for the RCPSP are given in Section 16.4. In Section 16.5, computational results are presented and discussed. Finally, Section 15.6 provides a summary of the paper and discusses future research ideas.

12.2 Literature review

The research literature for the RCPSP is quite large. We refer the readers to the review papers by Icmeli et al (1993), Ozdamar and Ulusoy (1995), Herroelen et al (1998), Brucker et al (1999), Hartmann and Kolisch (2000), Kolisch and Padman (2001), Kolisch and Hartmann (2005).

The various exact methods applied to the RCPSP can be classified into three categories: dynamic programming, zero-one programming and implicit enumeration with branch and bound. Pritsker et al (1969), Patterson and Huber (1974), Patterson and Roth (1976) proposed zero-one programming methods. Exact approaches based on implicit enumeration with branch and bound have been widely used: Davis and Heidorn (1971), Talbot and Patterson (1978), Christofides et al (1987), Demeulemeester and Herroelen (1992), Demeulemeester and Herroelen (1997), Brucker et al (1998), Mingozzi et al (1998), Dorndorf et al (2000). Blazewicz et al (1983) showed that the RCPSP is a generalization of the well-known job-shop-scheduling problem and is NP-Hard. While exact solution methods are able to solve smaller problems, heuristic and metaheuristic approaches are needed for larger problem instances.

Priority-rule based heuristics combine one or more priority rules and schedule-generation schemes (serial, parallel or both) in order to construct one or more schedules (Hartmann and Kolisch (2000)). If only one schedule is generated, it is called a single pass method and if more than one schedule is generated, it is called an X-pass (or multi-pass) method. Some of the well-known priority rules are LFT (Latest Finish Time), EST (Earliest Start Time) and MTS (Most Total Successor). Although, priority-rule based heuristics are easy to implement and fast in terms of the computational effort, they are not very effective with respect to the average deviation from the optimal solution. A variety of priority single-pass methods have been widely used to solve the RCPSP: Davis and Patterson (1975), Cooper (1976), Alvares-Valdes and Tamarit (1989), Boctor (1990), Ozdamar and Ulusoy (1994), Kolisch (1996a), Kolisch (1996b). Multi-pass methods can be categorized as multi-priority rule methods and sampling methods. Multi-priority rule methods combine the schedule generation scheme with a different priority rule at each iteration: Ulusoy and Ozdamar (1989), Boctor (1990), Thomas and Salhi (1998). Sampling methods use a serial generation scheme and a priority rule to obtain the first schedule. Then they bias the order obtained by the priority rule by using a random device: Cooper (1976), Alvares-Valdes and Tamarit (1989), Drexl (1991), Kolisch (1996a), Kolisch (1996b), Kolisch and Drexl (1996), Schirmer and Riesenberg (1998), Schirmer (2000).

Many metaheuristic methods, such as genetic algorithms (GA), simulated annealing (SA), tabu search (TS), and ant colonies (AC), have been applied to solve the RCPSP. Metaheuristics based on GA are the most common: Leon and Ramamoorthy (1995), Lee and Kim (1996), Hartmann (1998), Hartmann (2002), Alcaraz and Maroto (2001), Coelho and Tavares (2003), Hindi et al (2002), Toklu (2002), Valls et al (2003). Simulated annealing algorithms which can handle non-preemptive resource constrained project scheduling problem are presented by Boctor (1996), Cho and Kim (1997), Bouleimen and Lecocq (2003). Tabu search based metaheuristics are proposed by Pinson et al (1994) and Baar et al (1997), Nonobe and Ibaraki (2002) and Thomas and Salhi (1998). Merkle et al (2002) proposed an ant colony approach to the RCPSP.

In addition to applying these heuristics and metaheuristics, forward-backward improvement (FBI) steps are suggested by Tormos and Lova (2001), Tormos and Lova (2003) and Valls et al (2005). This step is also called double justification technique. In FBI, a given schedule is compressed by eliminating unnecessary pockets of slack on a Gantt Chart.

Surprisingly, neural network (NN) based techniques have not been applied to the RCPSP to the best of our knowledge. NN based approach has been applied to the job-shop scheduling problem (Foo and Takefuji (1988), and Sabuncuoglu and Gurgun (1996)), and the traveling salesman problem (Hopfield and Tank (1985)). Their approach is based on Hopfield networks. While this approach worked for smaller problem instances (up to 5x5), it failed to provide good solutions in reasonable time, for larger problem instances such as (10x10). Agarwal et al (2003) proposed a different kind of approach for using neural networks, called the AugNN approach, for solving task-scheduling problems. The performance of this alternative NN approach does not deteriorate with larger problem size.

12.3 Adaptive learning approach for serial schedule generation

As mentioned in Section 1, two types of schedule generation schemes are used in RCPSP, viz., serial and parallel. In the serial scheduling scheme, a priority list of activities is determined at time zero. This list is based on some heuristic such as latest finish time (LFT). The ordering of activities in a given priority list must, of course, follow precedence constraints, but is independent of the resource constraints. Given a priority list, activities are scheduled in the given order at the earliest possible clock time at which the precedence constraints are satisfied and the resources are available. This type of scheduling is similar to the permutation flow-shop scheduling in which the order of jobs is fixed a priori and scheduling occurs at the earliest possible clock time depending on resource availability and precedence constraints. Agarwal et al (2005) applied an adaptive learning approach to the permutation flow-shop problem. Due to the similarities between these two problems, we apply the ALA approach to the serial schedule generation.

The ALA is a non-deterministic local-search approach based on neural-networks principles. In this approach, processing times of activities are parameterized using a weight factor. The problem of optimally scheduling a given project is then posed as the problem of finding the optimal set of weights in the weight search space, similar to the way non-linear mapping functions are determined in neural networks. Reinforcement and backtracking techniques are applied as part of weight modification strategies.

Notation used:

A	:	Set of activities = $\{1, .., n\}$
A_j	:	j^{th} activity node, $j \in A$
z	:	Current iteration
PT_j	:	Processing time of activity j
W_j	:	Weight associated with the Activity A_j
WPT_j	:	Weighted processing time of Activity A_j
EST_j	:	Earliest start time of activity j
LST_j	:	Latest start time of activity j
Z_{max}	:	Max number of iterations
MS_z	:	Makespan in the z^{th} iteration
RF	:	Reinforcement factor
$TINI$	:	Tolerate iterations with no improvement
α	:	Learning rate
BMS	:	Best makespan
BW_j	:	Best weights

```
Step 1 : Initialization
         Initialize $\forall j \in A \quad W_j = 1$
         Initialize the iteration counter $z$ to 1.
Step 2 : Calculate weighted processing times
         Calculate $\forall j \in A \quad WPT_j = W_j * PT_j$
```

```
Step 3 : Determine priority list
         Determine the priority list using a heuristic such as
         earliest start time (EST) or latest finish time (LFT),
         where EST or LFT are calculated using $WPT_j$ instead of
         $PT_j$.
Step 4 : Determine makespan
         Find a feasible schedule using the priority list. In other
         words, schedule each activity at its earliest possible time
         given precedence and resource constraints. This
         schedule gives us the makespan $MS_z$.
Step 5 : Apply learning strategy and modify weights
         a. If $MS_z$ is the best makespan so far, save the current
            weights as best weights ($BW_j$) and the makespan as
            the best makespan ($BMS$).
         b. If $z = Z_{max}$, go to Step 7.
         c. If $z > 1$, and if an improvement occurs, reinforce the
            weights as follows:
            $(W_j)_z = (W_j)_z + RF * ((W_j)_z - (W_j)_{z-1})$.
            If no improvement occurs in this iteration, continue.
         d. If $z > TINI$ and if no improvement has occurred in
            the past TINI iterations then
            Set $W_j = BW_j$
         e. Modify the weights using the following strategy:
            Generate a random number RND between 0 and 1
            using uniform distribution.
            If RND $> 0.5$ then $(W_j)_{z+1} = (W_j)_z$ + RND*$\alpha$ *$P_j$
            If RND $<= 0.5$ then $(W_j)_{z+1} = (W_j)_z$ - RND*$\alpha$ *$P_j$
Step 6 : Next iteration.
         Increment $z$ by one and go to step 2.
Step 7 : Display Solution
         The BMS is the solution. Take the BWj and generate
         the schedule using the heuristic.
```

The learning rate (α) used in Step 5e determines the degree of weight change per iteration. A higher rate leads to a greater change and vice versa. One could therefore control the granularity of the search by varying α. The learning rate should neither be too low nor too high. A low α will slow down convergence and make it difficult to jump local minima, while a high α will render the search too erratic or volatile to afford convergence. With some empirical trial and error,. we found that a rate of 0.001 worked well for all the problems.

The reinforcement factor, RF, used in Step 5c is used to reinforce weights during iterations that show improved results. Intuitively, such reinforcement learning, common in neural networks, helps the search process by allowing the search to explore the newly discovered good neighborhood for a few extra iterations. Empirically, we found that an RF value of 2 gave better results than other RF values. Backtracking used in step 5d is used to backtrack to the previous best set of weights if no improvement has been found in a given number of iterations.

12.4 AUGNN framework for parallel schedule generation

In parallel schedule generation, the order in which the activities are going to be scheduled is not decided at time zero. The scheduling decisions are made on a clock timer, at times when activities can start and resources are available. Agarwal et al (2003) applied the AugNN approach for parallel schedule generation for the task scheduling problem which is a special case of RCPSP. In task scheduling, there is only one type of resource and each task needs only one unit of that resource type. We extend Agarwal et al's AugNN formulation of task scheduling problem to generate parallel schedules for RCPSP in which there are multiple resources and each activity needs multiple units of each resource. We describe the AugNN framework with the help of a simple RCPS problem shown in Figure 16.1. In this problem, there are six activities plus two zero-time dummy activities for the initial and final activities. There are three renewable resource types R1, R2 and R3. Each activity's duration and resource requirements are also shown in Figure 16.1. In this problem, the longest path (in terms of number of activities) has 5 activities – A1, A4, A6 or A2, A5, A6 plus the two dummy activities.

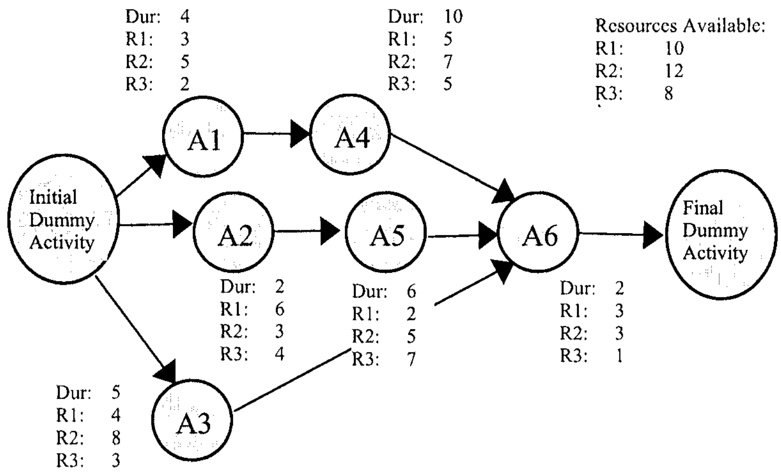


Figure 12.1. Example RCPS Problem

In the AugNN approach, the project graph of Figure 16.1 is framed as an n-layered neural network as shown in Figure 16.2. n, in this case is 8, which is 2(5-1), because 5 is the number of activities on the longest path. The set of activities at each level are placed in an activity layer. The p^{th} activity layer is p activities removed from the initial dummy activity. Each activity layer, except the dummy activity layers, is followed by a resource layer. The resource layer represents all the available resources. Input, activation and output functions for activity and resource layers are designed to capture the constraints of the problem. We will briefly describe the purpose of these functions.

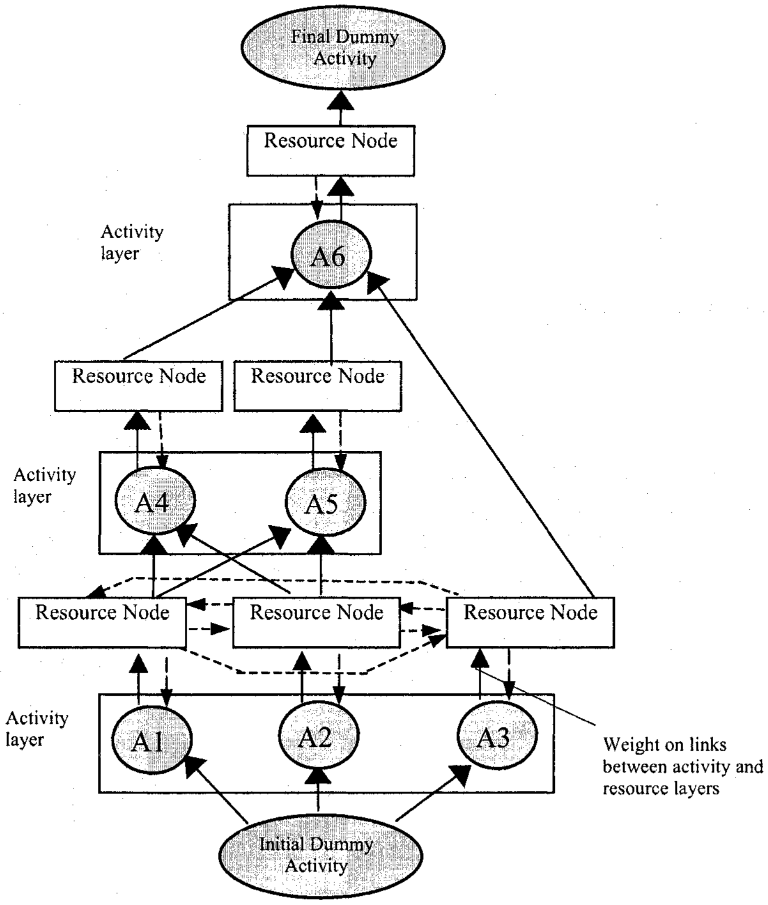


Figure 12.2. AugNN Representation of the Example Problem

Input Functions

The activity nodes get their input from the initial dummy node and resource layers. These functions are used to enforce the precedence constraints. When an activity node receives as many signals as the number of preceding activities, the activity is considered ready to be assigned.

The resource layer gets its input from the activity nodes preceding it. When it gets an input, it knows that the activity is ready to start and that resources, if available, can be assigned.

Activation Functions

The activation functions of the activity nodes maintain the state of each activity. The activation functions of the resource layer keep track of resource availability and assignment of resources to activities. The priority rule is applied through these activation functions.

Output Functions

The output function of the activity node sends a signal to the resource layer when it is ready to be assigned. The output function of the resource layer signals the end of the activity.

12.4.1 Mathematical Formulation and Algorithm Details

Notation

n	:	Number of activities
r	:	Number of resource types
A	:	Set of activities = $\{1,\ldots,n\}$
R	:	Set of resource types = $\{1,\ldots,r\}$
r_{ik}	:	Amount of resource k required by activity i, $k \in R$, $i \in A$
b_k	:	Total availability of resource type k
k	:	Current iteration
A_j	:	j^{th} activity node, $j \in A$
RL_j	:	Node for resource layer connected from $A_j, j \in A$
TS_j	:	Total number of successors of $A_j, j \in A$
$TRPT_j$	:	Total remaining processing time of $A_j, j \in A$
SLK	:	Slack for activity $A_j, j \in A$
ω_j	:	Weight on the link from A_j to resource layer
α	:	Learning coefficient
ε_k	:	Error in iteration k
I	:	Initial dummy activity node
F	:	Final dummy activity node

τ_j	:	Threshold value of A_j = Number of tasks immediately preceding A_j, $j \in A \cup F$
t	:	Elapsed time in current iteration
ST_j	:	Start time of activity j.
PT_j	:	Processing time of activity j
LST_j	:	Latest start time of activity j
LFT_j	:	Latest finish time of activity j
PR_j	:	Set of tasks that immediately precede task j, $j \in A \cup F$
SU_j	:	Set of tasks immediately succeeding task j, $j \in A$
Win_j	:	Winning status of A_j, $j \in A$
SCP	:	Set of tasks currently in process.

Following are all functions of elapsed time t:

$IA_j(t)$	:	Input function value of activity node j, $j \in I \cup A \cup F$
$IRLA_j(t)$	:	Input function value of resource layer from activity node A_j, $j \in A$
$IRLRL_{jk}(t)$	:	Input function value of resource layer j from other resource layers for each resource type k, $j \in A$, $k \in R$
$OA_j(t)$	:	Output function value of activity node j, $j \in I \cup A \cup F$
$ORLF_{jp}(t)$	:	Output of RL_j to activity node A_p in the forward direction, $j \in A$, $p \in SU_j$
$ORLR_j(t)$	:	Output of Resource layer RL_j to activity node A_j in reverse direction, $j \in A$
$ORLL_{jp}(t)$	:	Output of Resource layer RL_j to RL_p in lateral direction, $j,p \in A$, $j \neq p$
$\theta A_j(t)$	:	Activation function of activity node j, $j \in A$
$\theta RL_j(t)$	:	Activation function of Resource layer RL_j, $j \in T$
$assign_j(t)$	:	Activity j assigned at time t
$S(t)$	:	Set of activities that can start at time t. $S(t) = \{A_j \mid OA_j(t) = 1\}$
$RAv_k(t)$	:	Number of resources of type k available at time t

12.4.2 Preliminary Steps

1 Calculate the Lower Bound, which is the same as the critical path duration under infinite resource availability assumption.

2 Weights (ω_j) are initialized at 10.00. The value of 10 was arrived at after some computational experience. The value of the initial weights should be such that after subsequent modification to weights, the value should remain positive. The choice of the value of initial weight therefore also depends on the value of the learning coefficient used.

3 Calculate the threshold of each task τ_j. The threshold of activity j is defined as the number of tasks immediately preceding task j. The threshold value is used to determine when a task is ready to start.

12.4.3 AugNN Functions

The neural network algorithm can be described with the help of the learning strategy and the input functions, the activation functions and the output functions for the task nodes and the machine nodes.

12.4.3.1 Activity layer functions. Input functions, activation states and output functions are now explained for the nodes on the activity layer.

Input function
$\forall j \in A \cup F \quad IA_j(0) = 0$
$\forall j \in I$ *(the starting signal of the initial dummy node is 1)* $\quad IA_j(0) = 1$
$\forall q \in PR_j, j \in T \cup F \quad IA_j(t) = IA_j(t-1) + \sum_q ORLF_{qj}(t)$

IA_j helps to enforce precedence constraint. When IA_j becomes equal to τ_j, the activity can be assigned.

Activation function
Activity nodes' initial activation state (i.e. at t=0) is 1. $\forall j \in T$

$$\theta A_j(t) = \begin{cases} 1 & \text{if} \quad IA_j(t) < \tau_j \\ 2 & \text{if} \quad (\theta A_j(t-1) = 1 \vee 2) \wedge IA_j(t) = \tau_j \\ 3 & \text{if} \quad (\theta A_j(t-1) = 2 \vee 3) \wedge ORLR_j(t) < 0 \\ 4 & \text{if} \quad (\theta A_j(t-1) = 4 \vee (\theta A_j(t-1) = 3 \wedge ORLR_j(t) = 0) \end{cases}$$

Note: For the initial dummy node, $\tau_j = 1$
State 1 above implies that activity j is not ready to be assigned because input to activity j is less than its threshold τ. State 2 implies that activity j is ready to be assigned because its input equals its threshold. State 3 implies that the activity is in process because it is receiving a negative signal from the resource layer that it is currently being processed. State 4 implies that the activity is complete and the negative signal from the resource layer is no longer there.

Output function

$$OA_j(t) = \begin{cases} 1 \text{ if } \theta A_j(t) = 2 \\ 0 \text{ otherwise} \end{cases}$$

If an activity is ready to start but not assigned yet, it sends a unit signal to the resource layer.

F-Node

$$OA_j(t) = \begin{cases} t \text{ if } IA_F(t) = \tau_j \\ 0 \text{ otherwise} \end{cases}$$

The final node outputs the makespan t, the moment its threshold point is reached.

12.4.3.2 Resource layer functions. Input, activation and output functions of resource layers are now explained.

Input function

$\forall j \in A \quad IRLA_j(t) = OA_j(t) * \omega_j$

This is the weighted output from activity node j. Whenever it is positive, it means that the resources are being requested by activity j for assignment.

$\forall j \in A, k \in R \quad IRLRL_{jk}(t) = \sum_{p \,\varepsilon\, SCP} ORLL_{pjk}$

Activation function

Let $\chi_j(t) = IRL_j(t) * TaskHeuristicParameter_j$

Let $RAv_j(t)$: *Whether resources for activity j are available or not*

$$RAv_j(t) = \begin{cases} 1 & \text{if } \quad \forall k \in R \ (b_k - IRLRL_{jk} \geq r_{jk}) \\ 0 & \text{otherwise} \end{cases}$$

$$assign_j(t) = \begin{cases} 1 & \text{if} \quad RAv_j(t) = 1 \wedge \\ & \chi_j(t) = \max[\chi_j(t) | A_j \in S(t)] \wedge \forall j \in A \ \chi_j(t) > 0 \\ 0 & otherwise \end{cases}$$

The assignment takes place if the product of Input of the resource layer and the Heuristic dependent activity parameter is positive and highest and if the resources are available. The requirement for highest is what enforces the chosen heuristic.

TaskHeuristicParameter is a task parameter dependent on the chosen heuristic.

$$TaskHeuristicParameter = \begin{cases} TRPT & \text{for } TRPT \text{ heuristic} \\ LFT & \text{for } LFT \text{ heuristic} \\ EST & \text{for } EST \text{ heuristic} \\ EFT & \text{for } EFT \text{ heuristic} \\ LST & \text{for } LST \text{ heuristic} \\ RND & \text{for } RANDOM \text{ heuristic} \end{cases}$$

If $assign_j(t) = 1$, then $ST_j = t$.

If $|S(t)| > 1$ then $Win_j = 1$.

Resource layers' Initial Activation State (i.e. at $t = 0$) is 1. $\forall i \in M, j \in T$,

$$\theta RL_j(t) = \begin{cases} 1 & \text{if} \quad RAv_j(t) = 1 \\ 2 & \text{if} \quad \theta RL_j(t-1) = 1 \vee \theta RL_j(t) = 1) \wedge assign_j(t) = 1 \\ 3 & \text{if} \quad (\theta RL_j(t-1) = 2 \vee 3) \wedge t < ST_j + PT_j : \\ 4 & \text{if} \quad \theta RL_j(t-1) = 3 \wedge t = ST_j + PT_j \end{cases}$$

State 1 implies that the resources are available. State 2 implies that the resources are busy and that they were just assigned. State 3 implies that the resources are busy and state 4 implies that the resources were just released.

Output function

$$ORLF_{jp}(t) = \begin{cases} 1 & \text{if } \theta RL_j(t) = 4 \\ 0 & \text{if } \theta RL_j(t) = 1,2,3 \end{cases} \quad j \in A, p \in SU_j$$

$$ORLR_j(t) = \begin{cases} -1 & \text{if } \theta RL_j(t) = 2,3 \\ 0 & \text{if } \theta RL_j(t) = 1,4 \end{cases} \quad j \in A$$

$$ORLL_{pjk}(t) = \begin{cases} r_{jk} & \text{if } \theta RL_j(t) = 2,3 \\ 0 & \text{if } \theta RL_j(t) = 1,4 \end{cases} \quad j, p \in A, k \in R, p \neq j$$

The output of F represents the makespan and the $assign_j(t)$ gives the schedule. If a resource is either assigned or released during a certain time unit, all functions need to be recalculated without incrementing the time period.

12.4.4 Learning Strategy

A learning strategy is required to modify the weights. In this study we used the following learning strategy:

Winning activities:

If $\forall j \in A \quad Win_j = 1$ then $(\omega_j)_{k+1} = (\omega_j)_k - \alpha * TaskParameter_j * \varepsilon_k$

Non-winning activities:

If $\forall j \in A \quad Win_j = 0$ then $(\omega_j)_{k+1} = (\omega_j)_k + \alpha * TaskParameter_j * \varepsilon_k$

In addition to these weight changes in each iteration, we propose two additional features that govern learning, namely, reinforcement and backtracking. These features are explained here briefly.

Reinforcement:

Neural Networks use the concept of positive reinforcement of weights if the network performs well. We implement this idea of reinforcement by implementing the following rule. If in a particular iteration the makespan improves, the weight changes of that iteration with respect to the previous iteration are magnified by a factor called the reinforcement factor (RF).

$$(\omega_{ik})_{k+1} = (\omega_{ik})_{k+1+RF*}((\omega_{ik})_{k+1} - (\omega_{ik})_k)$$

Backtracking:

Sometimes it is possible to not obtain any improvement over several iterations. When this happens, it is best to abandon that search path and start over from the previous best solution weights. We can parameterize how many iterations of no improvement to tolerate. This backtracking technique was part of our learning strategy. In order to do this, we store the set of weights corresponding to the best solution obtained so far and revert back to it whenever solution does not improve for some iterations.

12.4.5 End of iteration routines

1 Calculate the gap which is the difference between obtained makespan and the lower bound

2 Store the best solution so far.

3 If the lower bound is reached, stop the program.

4 If the number of iterations exceeds a certain specified number, such as 1000 or 5000, stop the program.

5 If continuing with the next iteration, modify weights using the learning strategy. Apply backtracking and reinforcement, whenever necessary.

12.5 Computational experiments and results

We now present the result of the computational tests and compare them with the best published algorithms. The HNA approach algorithms were coded in Visual Basic 6.0 and run on a Celeron 2300 MHz personal computer. Well known benchmark problem instance sets (Kolisch et al (1995) and Kolisch and Sprecher (1996)) are used to evaluate the algorithm (PSPLIB, http://www.bwl.uni-kiel.de /Prod/psplib/ index.html). The sets J30 and J60 consists of 480 problem instances with four resource types and 30 and 60 activities, respectively. The set J120 consists of 600 problem instances with four resource type and 120 activities.

In our implementation, the learning coefficient α is set to 0.001 and the weights are initialized at 10. An initial solution is generated using a priority rules such as LFT or EST. The weights are modified after each iteration, using the learning strategy. The stopping criterion is to stop if the solution is equal to the lower bound or if a predetermined number of maximum schedules is reached. We set the maximum number of schedules to 1000 and 5000.

Tables 16.1 through 16.3 display the results obtained by our algorithm and other tested heuristics for 1,000 and 5,000 schedules, respectively. In these

Table 12.1. Percent Average Deviations from the Optimum Solution: Comparative Results for the J30 Problems

			# of Schedules	
Algorithm	**SGS**	**Reference**	**1,000**	**5,000**
Sampling - LFT – FBI	both	Tormos and Lova (2003)	0.23	0.14
GA - FBI	both	Alcaraz et al (2004)	0.25	0.06
HNA - FBI	**both**	**this paper**	**0.25**	**0.11**
Sampling - LFT - FBI	both	Tormos and Lova (2001)	0.25	0.15
GA – hybrid, FBI	serial	Valls et al (2003)	0.27	0.06
Scatter Search – FBI	serial	Debels et al (2004)	0.27	0.11
GA – FBI	serial	Alcaraz and Maroto (2001)	0.33	0.12
GA – FBI	serial	Valls et al (2005)	0.34	0.20
GA - Self adapting	both	Hartmann (2002)	0.38	0.22
SA - Activity List	serial	Bouleimen and Lecocq (2003)	0.38	0.23
TS - Activity List	serial	Nonobe and Ibaraki (2002)	0.46	0.16
Sampling – FBI	serial	Valls et al (2005)	0.46	0.28
GA - Activity List	serial	Hartmann (1998)	0.54	0.25
Adaptive Sampling	both	Schirmer (2000)	0.65	0.44
GA	serial	Coelho and Tavares (2003)	0.74	0.33
Adaptive Sampling	both	Kolisch and Drexl (1996)	0.74	0.52
Sampling – Global	serial	Coelho and Tavares (2003)	0.81	0.54
TS		Baar et al (1997)	0.86	0.44
GA – Random Key	serial	Hartmann (1998)	1.03	0.56
GA – Priority Rule	serial	Hartmann (1998)	1.38	1.12
Sampling – WCS	parallel	Kolisch (1996b)	1.40	1.28
Sampling - LFT	parallel	Kolisch (1996b)	1.40	1.29
Sampling – random	serial	Kolisch (1995)	1.44	1.00
Sampling – random	parallel	Kolisch (1995)	1.77	1.48
GA		Leon and Ramamoorthy (1995)	2.08	1.59

tables we present the type of heuristics, the type of schedule generation scheme used, the authors of each heuristic and the average deviation from the critical path based lower bound (from the optimal solution for J30 instances) for 1000 and 5000 schedules, respectively. In each table, the heuristics are sorted according to descending performance with respect to 1000 schedules.

Table 16.1 presents the percentage deviations from the optimal makespan for the instance set J30 in which all problem instances have been solved to optimality by Demeulemeester and Herroelen (1997) branch and bound procedure. Our algorithm solved 448 out of 480 problems to optimality and the average deviation from the optimal solution is just 0.25 and 0.11 percent for 1000 schedules and 5000 schedules, respectively.

Table 12.2. Percent Average Deviations from the Critical-Path-Based Lower Bound: Comparative Results for the J60 Problems

			# of Schedules	
Algorithm	**SGS**	**Reference**	**1,000**	**5,000**
GA – hybrid, FBI	serial	Valls et al (2003)	11.56	11.10
HNA - FBI	**both**	**this paper**	**11.72**	**11.39**
Scatter Search – FBI	serial	Debels et al (2004)	11.73	11.10
GA - FBI	both	Alcaraz et al (2004)	11.89	11.19
Sampling - LFT – FBI	both	Tormos and Lova (2003)	12.04	11.72
Sampling - LFT - FBI	both	Tormos and Lova (2001)	12.11	11.82
GA – FBI	serial	Valls et al (2005)	12.21	11.27
GA - Self adapting	both	Hartmann (2002)	12.21	11.70
GA – FBI	serial	Alcaraz and Maroto (2001)	12.57	11.86
GA - Activity List	serial	Hartmann (1998)	12.68	11.89
Sampling – FBI	serial	Valls et al (2005)	12.73	12.35
SA - Activity List	serial	Bouleimen and Lecocq (2003)	12.75	11.90
Adaptive Sampling	both	Schirmer00	12.94	12.59
TS - Activity List	serial	Nonobe and Ibaraki (2002)	12.97	12.18
GA	serial	Coelho and Tavares (2003)	13.28	12.63
GA – Priority Rule	serial	Hartmann (1998)	13.30	12.74
Adaptive Sampling	both	Kolisch and Drexl (1996)	13.51	13.06
Sampling - LFT	parallel	Kolisch (1996b)	13.59	13.23
Sampling – WCS	parallel	Kolisch (1996b)	13.66	13.21
Sampling – Global	serial	Coelho and Tavares (2003)	13.80	13.31
TS		Baar et al (1997)	13.80	13.48
GA		Leon and Ramamoorthy (1995)	14.33	13.49
GA – Random Key	serial	Hartmann (1998)	14.68	13.32
Sampling – random	parallel	Kolisch (1995)	14.89	14.30
Sampling – random	serial	Kolisch (1995)	15.94	15.17

For J60 problems set, some of the optimal solutions are not known, so we measure the average percentage deviation from the critical-path based lower bound for comparison purposes. Table 16.2 summarizes the results for J60 test instances. 295 out of 480 instances are solved to critical path based lower bound. The average deviation from the critical path based lower bound is 11.72 and 11.39 percent for 1000 and 5000 schedules, respectively.

Table 12.3. Percent Average Deviations from the Critical-Path-Based Lower Bound: Comparative Results for the J120 Problems

			# of Schedules	
Algorithm	**SGS**	**Reference**	**1,000**	**5,000**
GA – hybrid, FBI	serial	Valls et al (2003)	34.07	32.54
HNA - FBI	**both**	**this paper**	**34.94**	**34.57**
Scatter Search – FBI	serial	Debels et al (2004)	35.39	33.24
GA – FBI	serial	Valls et al (2005)	35.98	35.30
Sampling - LFT – FBI	both	Tormos and Lova (2003)	35.98	35.30
Sampling - LFT - FBI	both	Tormos and Lova (2001)	36.32	35.30
GA - FBI	both	Alcaraz et al (2004)	36.53	33.91
GA - Self adapting	both	Hartmann (2002)	37.19	35.39
Sampling – FBI	serial	Valls et al (2005)	38.21	37.47
GA – FBI	serial	Alcaraz and Maroto (2001)	39.36	36.57
GA - Activity List	serial	Hartmann (1998)	39.37	36.74
Sampling - LFT	parallel	Kolisch (1996b)	39.60	38.75
Sampling – WCS	parallel	Kolisch (1996b)	39.65	38.77
Adaptive Sampling	both	Schirmer (2000)	39.85	38.70
GA – Priority Rule	serial	Hartmann (1998)	39.93	38.49
GA	serial	Coelho and Tavares (2003)	39.97	38.41
TS - Activity List	serial	Nonobe and Ibaraki (2002)	40.86	37.88
Sampling – Global	serial	Coelho and Tavares (2003)	41.36	40.46
Adaptive Sampling	both	Kolisch and Drexl (1996)	41.37	40.45
SA - Activity List	serial	Bouleimen and Lecocq (2003)	42.81	37.68
Sampling – LFT	Serial	Kolisch (1996b)	42.84	41.84
GA		Leon and Ramamoorthy (1995)	42.91	40.69
Sampling – random	parallel	Kolisch (1995)	44.36	43.05
GA – Random Key	serial	Hartmann (1998)	45.82	45.25
Sampling – random	serial	Kolisch (1995)	49.25	47.61

Table 16.3 summarizes the results for J120 set. 155 out of 600 problems matched the critical path based lower bound. The average deviations are 34.94 and 34.57 percent, respectively, for 1000 and 5000 schedules.

12.6 Conclusions

We proposed, developed and tested a new metaheuristic approach based on the principles of neural networks. Augmented-neural-network approach was used for parallel schedule generation and adaptive-learning approach for serial schedule generation scheme. We called this approach the hybrid neural approach (HNA). To the best of our knowledge, this is the first time that neural-networks based metaheuristics have been applied to the RCPSP. So, research in this approach is still in its infancy. We tested this approach on some well-known RCPSP benchmark problem instances in the literature. The computational results are very encouraging as they compare very well with some of the best results in the literature from techniques such as tabu search, simulated annealing, genetic algorithms and scatter search. The approach, in spite of being relatively new, gave very good results, and therefore appears to be very promising and worthy of further exploration.

Future research may focus on developing some hybrid approaches involving the HNA approach and some of the other successful approaches such as genetic algorithms and scatter search, to further improve the results. This new approach should also be applied to multi-mode resource constrained project scheduling problems with renewable and non-renewable resources.

References

Agarwal, A., Jacob, V.S. and Pirkul, H. (2003). Augmented neural networks for task scheduling, *European Journal of Operational Research*, 151(3):481–502.

Agarwal, A., Colak, S. and Eryarsoy, E. (2005). Improvement heuristic for the flow-shop scheduling problem: an adaptive-learning approach, *European Journal of Operational Research*, 169(3):801–815.

Alcaraz, J. and Maroto, C. (2001). A robust genetic algorithm for resource allocation in Project Scheduling, *Annals of Operations Research*. 102:83–109.

Alcaraz, J., Maroto, C. and Ruiz, R. (2004). Improving the performance of genetic algorithms for the RCPS problem, *Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pp. 40–43.

Alvares-Valdes, R. and Tamarit, J.M. (1989). Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis, in: *Advances in Project Scheduling*, R. Slowinski, J. Weglarz (Eds.), Elsevier, Amsterdam, pp. 113–134.

Baar, T., Brucker, P. and Knust, S. (1997). Tabu search algorithms for resource-constrained project scheduling problems, in: *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimisation*, S. Voss, S. Martello, I. Osman, C. Roucairol (Eds.), Kluwer, pp. 1–18.

Blazewicz, J., Lenstra, J.K. and Rinnooy Kan A.H.G. (1983). Scheduling projects to resource constraints: classification and complexity, *Discrete Applied Mathematics*. 5:11–24.

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. (1996). An adaptation of the simulated annealing algorithm for solving resource-constrained project scheduling problems, *International Journal of Production Research*. 34:2335–2351.

Bouleimen, K. and Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, *European Journal of Operational Research*. 149:268–281.

Brucker, P., Knust, S., Schoo, A. and Thiele, O. (1998). A branch & bound algorithm for the resource-constrained project scheduling problem, *European Journal of Operational Research* 107(2):272–288.

Brucker, P., Drexl, A., Mohring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: notation, classification, models, and methods, *European Journal of Operational Research*. 112(1):3-41.

Christofides, N., Alvarez-Valdes, R. and Tamarit, J.M. (1987). Project scheduling with resource constraints: a branch-and-bound approach, *European Journal of Operational Research* 29(2):262–273.

Cho, J.H. and Kim, Y.D. (1997). A simulated annealing algorithm for resource-constrained project scheduling problems, *Journal of the Operational Research Society*. 48:736–744.

Coelho, J. and Tavares, L. (2003). Comparative analysis of meta–heuricstics for the resource constrained project scheduling problem, Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal.

Cooper, D.F. (1976). Heuristics for scheduling resource–constrained projects: An experimental investigation, *Management Science* 22:1186–1194.

Davis, E.W. and Heidorn, G.E. (1971). An algorithm for optimal project scheduling under multiple resource constraints, *Management Science* 17:803–816.

Davis, E.W. and Patterson, J.H. (1975). A comparison of heuristic and optimum solutions in resource-constrained project scheduling, *Management Science*. 21:944–955.

Debels, D., De Reyck, B., Leus, R. and Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta-heuristic for project scheduling, *European Journal of Operational Research* 169(2):638–653.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problems, *Management Science*. 38(12):1803–1818.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem, *Management Science* 43(11):1485–92.

Dorndorf, U., Pesch, E. and Phan-Huy, T. (2000). A branch-and-bound algorithm for the resource-constrained project scheduling problem, *Mathematical Methods of Operations Research*, 52:413-439.

Drexl, A. (1991). Scheduling of project networks by job assignment, *Management Science.* 37:1590-1602.

Foo, Y.P.S. and Takefuji, Y. (1988). Stochastic neural networks for solving job-shop scheduling, *Proceedings of Joint International Conference on Neural Networks Vol. 2,* pp. 275-290.

Hartmann, S. (1998). A competitive genetic algorithm for the resource-constrained project scheduling, *Naval Research Logistics*, 45:733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints, *Naval Research Logistics.* 49:433–448.

Hartmann, S. and Kolisch, R. (2000). Experimental evaluation of state–of–the–art heuristics for the resource–constrained project scheduling problem, *European Journal of Operational Research* 127:394–407.

Herroelen, W., Demeulemeester, E. and De Reyck, B. (1998). Resource- constrained project scheduling: A survey of recent developments, *Computers & Operations Research* 25(4):279-302.

Hindi, K. S., Yang H. and Fleszar, K. (2002). An evolutionary algorithm for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation.* 6:512–518.

Hopfield, J.J. and Tank, D.W. (1985). Neural computation of decisions in optimization problems, *Biological Cybernetics.* 52:141-152.

Icmeli, O., Erenguc, S.S. and Zappe, C.J. (1993). Project scheduling problems: A survey, *International Journal of Operations & Production Management* 13(11):80-91.

Kolisch, R. (1995). *Project scheduling under resource constraints: efficient heuristics for several problem classes*, Physica, Heidelberg, Germany.

Kolisch, R. (1996a). Efficient priority rule for the resource-constrained project scheduling problem, *Journal of Operations Management.* 14(3):179–192.

Kolisch, R. (1996b). Serial and parallel resource–constrained project scheduling methods revisited: theory and computation, European *Journal of Operational Research* 90:320–333.

Kolisch, R. and Drexl, A. (1996). Adaptive search for solving hard project scheduling problems, *Naval Research Logistics.* 43:23–40.

Kolisch, R. and Hartmann, S. (2005). Experimental investigation of heuristics for resource–constrained project scheduling: An update, *European Journal of Operational Research*, forthcoming.

Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling, OMEGA. 29:249–272.

Kolisch R. and Sprecher, A. (1996). PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216.

Kolisch, R., Sprecher, A. and Drexl, A. (1995). Characterisation and generation of a general class of resource-constrained project scheduling problem, *Management Science*, 41(10): 1693–1703.

Lee, J.K. and Kim, Y.D., 1996, Search heuristics for resource-constrained project scheduling, *Journal of the Operational Research Society*. 47:678–689.

Leon, V.J. and Ramamoorthy, B. (1995). Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling, *OR Spektrum*. 17:173–182.

Mingozzi, A. , Maniezzo, V., Ricciardelli, S. and Bianco, L. (1998). An exact algorithm for project scheduling with resource constraints based on new mathematical formulation, *Management Science,* 44(5):714-29.

Merkle, D., Middendorf, M. and Schmeck H. (2002). Ant colony optimization for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation*. 6:333–346.

Nonobe, K. and Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: *Essays and Surveys in Metaheuristics*, C. C. Ribeiro and P. Hansen, eds, Kluwer Academic Publishers, pp. 557–588.

Ozdamar, L. and Ulusoy, G., (1994). A local constraint based analysis approach to project scheduling under general resource constraints, *European Journal of Operational Research*. 79:287–298.

Ozdamar, L. and Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem, *IIE Transactions*. 27:574-586.

Patterson, J.H. and Huber, W.D. (1974). A horizon-varying, zero-one approach to project scheduling, *Management Science* 20:990–998.

Patterson, J.H. and Roth, G.W., 1976, Scheduling a project under multiple resource constraints: a zero-one programming approach, *AIIE Transactions*. 8:449–55.

Pinson, E., Prins, C. and Rullier, F. (1994). Using tabu search for solving the resource-constrained project scheduling problem, in: *Proceedings of the 4th International Workshop on Project Management and Scheduling*, Leuven, Belgium, pp. 102–106.

Pritsker, A.A.B., Watters, L.J. and Wolfe, P.M. (1969). Multiproject scheduling with limited resources: a zero-one programming approach, *Management Science* 16: 93–107.

Sabuncuoglu, I. and Gurgun, B. (1996). A neural network model for scheduling problems, *European Journal of Operational Research* 93:288-299.

Schirmer, A. and Riesenberg, S. (1998). Class-based control schemes for parameterized project scheduling heuristics, Manuskripte aus den Instituten für Betriebswirtschaftslehre 471, Universität Kiel, Germany.

Schirmer, A. (2000). Case–based reasoning and improved adaptive search for project scheduling, *Naval Research Logistics*.47:201–222.

Talbot, F.B. and Patterson, J.H. (1978). An efficient integer programming algorithm with network cuts for solving resource constrained scheduling problems, *Management Science*, 24(11):1163–74.

Thomas, P. R. and Salhi, S. (1998). A tabu search approach for the resource constrained project scheduling problem, *Journal of Heuristics*, 4:123–139.

Toklu, Y.C. (2002). Application of genetic algorithms to construction scheduling with or without resource constraints, *Canadian Journal of Civil Engineering*. 29:421-429.

Tormos, P. and Lova, A., 2001, A competitive heuristic solution technique for resource constrained project scheduling, *Annals of Operations Research*. 102:65–81.

Tormos, P. and Lova, A. (2003). An efficient multi-pass heuristic for project scheduling with constrained resources, *International Journal of Production Research*. 41(5):1071–1086.

Ulusoy, G. and Ozdamar, L. (1989). Heuristic performance and network/ resource characteristics in resource-constrained project scheduling, *Journal of the Operational Research Society*. 40:1145–1152.

Valls, V., Ballestin, F. and Quintanilla, M. S. (2003). A hybrid genetic algorithm for the RCPSP, Technical report, Department of Statistics and Operations Research, University of Valencia.

Valls, V., Ballestin, F. and Quintanilla M. S. (2005). Justification and RCPSP: A technique that pays, *European Journal of Operational Research*. 165(2):375-386.

III

APPLICATIONS

Chapter 13

SELECTION AND SCHEDULING OF PHARMACEUTICAL RESEARCH PROJECTS

Rainer Kolisch
Chair of Technology–based Services & Operations Management, TUM Business School
Technical University of Munich, Germany
rainer.kolisch@wi.tum.de

Konrad Meyer
A.T. Kearney, Frankfurt, Germany
konrad.meyer@atkearney.com

Abstract This paper deals with the lead optimization phase of pharmaceutical research where a number of leads (molecules as a basis for potential drugs) are processed by different departments in order to optimize their biochemical characteristics. We depict each lead as a project and model the problem as a static multi–project selection and scheduling problem under resource constraints with the objective to maximize the weighted work performed. For solving the problem we propose two heuristics. We assess their performance in a computational study and we point out one dominant method. Furthermore we show the impact of problem parameters such as the extend to which projects can be crashed.

Keywords: Pharmaceutical R&D, Lead Optimization, Multi–Mode Resource–Constrained Project Scheduling, Heuristics.

13.1 Introduction

A vital task of pharmaceutical companies is the research and development of new drugs. The latter is characterized by long lead times, expensive and thus scarce resources, a high attrition rate, and a strong time–based competition due to the fact that the company which first gets a new compound patented will have a time–limited monopoly to produce and market the associated drug. In this context it is crucial to select the right research projects and to schedule them in a proper manner in order to generate value.

The focus of this paper is twofold. First, we model the problem of selection and scheduling of pharmaceutical research projects. The problem arises in the lead optimization phase of a major European pharmaceutical company. We model it as a multi–project resource–constrained project selection and scheduling problem. Second, we propose two heuristic solution procedures. The heuristic methods are experimentally evaluated on a set of benchmark instances which has been generated based on real world data.

The paper is organized as follows: In Section 13.2 we provide a problem description and a brief literature review. Section 13.3 is devoted to the development of the integer programming model. The two heuristics are proposed in Section 13.4 and the experimental test and its results are reported in Section 13.5. Section 13.6 summarizes.

13.2 Problem Description

13.2.1 Drug Research and Development Process

The drug research and development process can be depicted by seven phases (cf. Figure 13.1). The first three phases are considered as research, the next three phases are considered as development, and the final phase is the marketing phase. The first six phases may take up to 15 years. In the sequel we give a brief description of the phases.

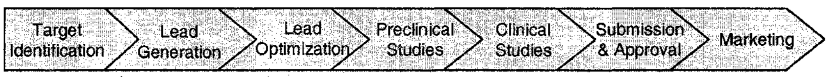


Figure 13.1. Drug Research and Development Process

Target Identification and Lead Generation Starting point of the drug research and development process is a disease for which the company wants to discover and develop a new drug. In order to cure the disease researchers look first for targets, biological elements of the human body which play an important role in the progression of the disease or its symptoms. The target identification phase comprises all activities concerned with finding such targets.

Within the next phase, lead generation, the goal is to find chemical compounds (molecules) which act on the target so that the disease can be treated. By using high-throughput technologies a large number of compounds, i.e. 60,000, is automatically generated and tested within a time span of two weeks. Molecules which show promising characteristics are called leads (because they will lead the way to new drugs).

Lead Optimization and Preclinical Studies The task of lead optimization is to optimize the characteristics of a lead in terms of its effectiveness to cure the disease and its safety profile for the human body. Lead optimization is an iterative and evolving process where leads are altered and tested. An optimized lead is termed candidate because it is a candidate for a new drug.

The lead optimization is the last research phase and the preclinical studies are the first development phase. Whereas in lead optimization testing is done only in vitro and by computational simulation studies (in silico), the preclinical phase does testing with animals (in vivo). The goal is to obtain enough information and reliability in order to proceed to human testing. In order to ensure the exclusive right to produce and market drugs which are based on the compound, a patent application for the latter is done during the phase of the preclinical studies or the succeeding subphase I of the clinical studies (cf. Schöffski et al (2002) p. 231).

Clinical Studies Candidate testing on humans is subject to the clinical studies which are divided roughly into three subphases.

Subphase I studies involve a small number (20 to 80) of healthy volunteers and are conducted to determine dosing levels and assess the safety, tolerability, dose response and metabolic properties of the compound in humans.

In subphase II studies the drug is administered to a larger number (50 to 500) of subjects. Phase II studies confirm the drug's safety profile in patients diagnosed with the disease being studied.

Subphase III studies are much larger in scale, and gather additional information about the drug's safety and effectiveness in the intended patient population. Depending on the therapeutic area, thousands of patients may be enrolled in studies that compare the drug being tested to one or more currently available therapies. The objective is to show statistical superiority via either improved efficacy or safety over current treatments. This critical endpoint is needed to obtain regulatory approval to market the drug.

Submission and Approval The data collected from the preclinical and clinical studies are compiled into reports for review and approval by governmental regulatory agencies such as the European Agency for the Evaluation of Medicinal Products (EMEA) and the Food and Drug Administration (FDA) in the US.

Marketing Once the application is approved, the company is allowed to market the drug. Until the end of the submission and approval phase the project will cause solely cost, while it will generate positive cash flows in the marketing phase.

13.2.2 Cycle Concept of Lead Optimization

The lead optimization phase is characterized by an iterative and evolving process where variants of the leads are altered and tested in order to optimize their characteristics.

At the start of the process, a single lead comes from the lead generation phase and is altered into 20 to 30 variants in the chemistry department. The variants are then processed in one lot in a number of different departments. The compatibility of the variants for the human body is tested by the early Administration, Distribution, Metabolism, Kinetics, Toxicology (eADMET) department. The effectiveness of the variants is tested in different subsections of the Disease Groups department. All testing is done in parallel by the use of copies of the variants. On account of the test results the best variant of the lead is selected and again altered to a number of variants in the chemistry department. The cycle is repeated for about 15 iterations until the lead has been optimized and a candidate has been found.

13.2.3 Organizational Frame and Decision Problem

A separate organizational function of the pharmaceutical company is responsible for the resources of each phase. The managers of this function team up with the managers of the projects to form a research committee which decides every three months on the selection or the termination of the projects and on the scheduling of the projects. Selection and scheduling is done subject to the limited availability of scarce resources. The overall objective of the company is to generate value. This objective is broken down to the functions. Due to organizational and management issues, the research committee of the lead optimization phase has the objective to maximize the weighted work performed by their function within the next quarter. Planning is done on the basis of a rolling horizon. In Section 13.3.1.3 we will detail how work is weighted in order to be linked to the overall objective of value generation.

13.2.4 Brief Literature Review

A number of contributions treat aspects of the problem described above. Venkatraman and Venkatraman (1995) employ a spreadsheet approach to consider the problem of product obsolescence when selecting and scheduling R&D projects. Luh et al (1999) treat the problem of scheduling a project with an uncertain number of cycles. Kolisch et al(2003) consider the problem of minimizing the makespan of a single lead–project. They propose a mixed–integer program and a simple priority rule–based heuristic. A considerable number of articles suggest the selection of R&D projects based on zero–one programming models: E.g. Taylor et al (1982) propose a non–linear multi–criteria integer programming model while Fox et al (1984) model project interactions whereas Heidenberger, K. (1996) considers a dynamic situation under risk. Project selection and scheduling is performed by Coffin and Taylor (1996) employing a filtered beam search approach. Finally, Viswanadham and Narahari (2001) propose a queuing network model in order to model the entire drug research and development process.

To the best of our knowledge there has been no contribution made for selecting and scheduling pharmaceutical research projects in the lead optimization phase by employing concepts of resource–constrained project scheduling.

13.3 Integer Programming Formulation

We model the problem of selection and scheduling of pharmaceutical research projects within the lead optimization phase as a zero–one optimization problem which is based on the concepts of multi–mode resource–constrained project scheduling with deadlines (cf. Neumann et al (2003)). We depict the processing of one lead as a single project, the set of all projects as a multi–project network, and the chemistry and eADMET departments as well as the subsections of the Disease Groups departments as renewable resources.

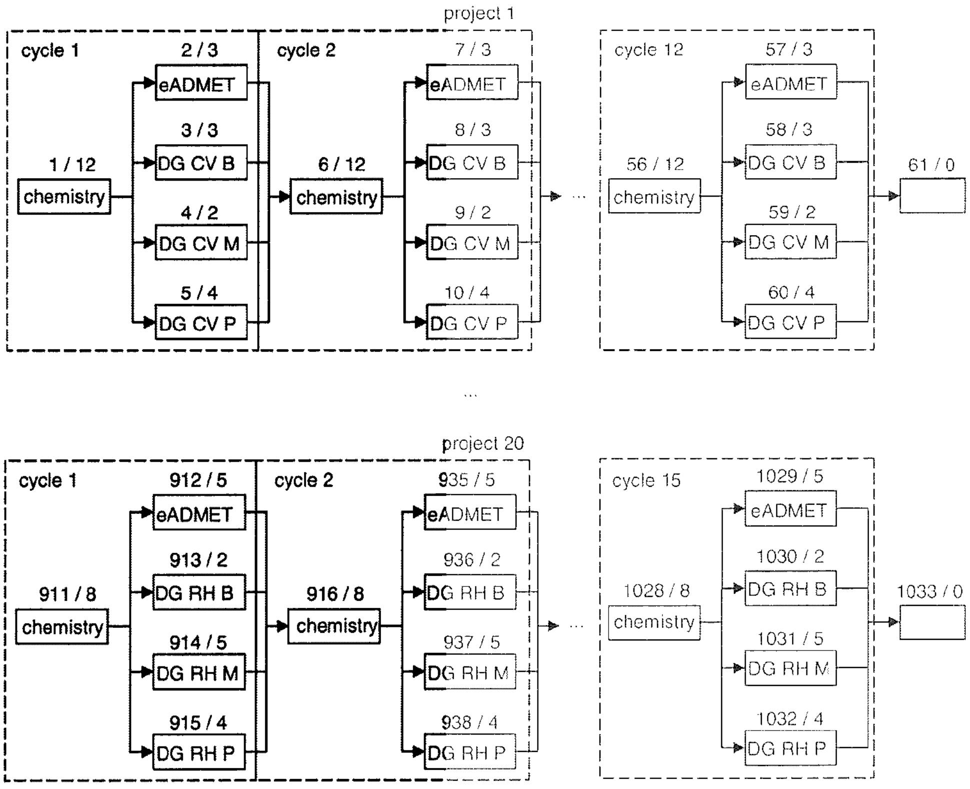


Figure 13.2. Multi–Project Network

13.3.1 Model Parameters

13.3.1.1 Projects and Resources. Let $\mathcal{P}$ be the set of lead optimization projects which are available. Projects have been either just created in the lead generation phase or are already in progress in the lead optimization phase. Each project $j \in \mathcal{P}$ consists of a set of activities $\mathcal{V}_j$ which depicts the tasks necessary to optimize the lead. Due to the cyclical structure described in Section 13.2.2, each project has the following special network structure (cf. Figure 13.2 where a box corresponds to an activity): The first cycle, and thus the project, starts with one activity which is processed in the chemistry department where the lead is altered to a number of variants. A number of parallel activities are immediate successors. One successor activity is processed in the eADMET department where the leads are tested w.r.t. tolerability for the human body. The remaining successor activities are processed in different subsections of the Disease Groups department where the leads are tested in terms of effectiveness. The organization of the Disease Groups department into sections and subsections will be detailed below. The single start activity and the parallel successor activities define the first cycle. This cycle is repeated for a given number of times, usually 10 to 15. At the end of the last cycle we add a dummy end activity which is the milestone for the end of the lead optimization project. The activities of the project are labeled consecutively. The set of all activities is denoted with $\mathcal{V}$ and the set of all precedence relations is denoted with $\mathcal{E}$. Note that there are no precedence relations between activities of different projects.

Due to a management policy projects have to be finished within a maximum time span where the lead generation is counted as the start of the project. By the time the project enters the lead optimization phase the lead generation has already been accomplished. Hence, within the lead optimization phase the maximum project duration becomes a deadline for the finish time of the project. The deadline of project j will be denoted with $\overline{d}_j$.

The resources of the lead optimization function are chemistry, eADMET and the 9 resources of the Disease Groups department. The set of all resources is denoted with $\mathcal{R}$. Table 13.1 gives an overview of the resources, their numbers k, and their capacity R_k measured in the number of available laboratory units per week. All resources are renewable (cf. Brucker et al (1999)) , i.e. their capacity is available anew for each period (week). As can be seen, the 9 Disease Groups resources are differentiated with respect to the field of therapy (Sections Cardiovascular, Metabolic, and Rheumatic) and the type of research (Subsections Biochemistry, Molecularbiology, and Pharmacology).

13.3.1.2 Activities and Modes. Activity i has to be processed on exactly one resource $k(i)$. The work content of activity i is r_i units. i and r_i are given in Figure 13.2 as i/r_i above each activity rectangle and $k(i)$ is written

Table 13.1. Resources

Department	Section	Subsection	R_k	k
Chemistry			75	1
eADMET			21	2
Disease Groups	Cardiovascular	Biochemistry	5	3
		Molecularbiology	7	4
		Pharmacology	9	5
	Metabolic	Biochemistry	6	6
		Molecularbiology	4	7
		Pharmacology	7	8
	Rheumatic	Biochemistry	8	9
		Molecularbiology	5	10
		Pharmacology	7	11

inside the rectangle. E.g., activity 1 has to be processed by resource 1 (chemistry) and the work content is 12 laboratory week units. Note that corresponding activities in each cycle of one project do request the same resource and do have the same work content. E.g., activity $3, 8, \ldots, 58$ of project 1 do all request resource 5 (DG CV B) and have a work content of 3 laboratory week units. The processing of the activity can be done in a number of different (discrete) ways. Each way is termed mode (cf. Patterson et al (1990) and Talbot (1982)). The set of all modes which are available for activity i is denoted with $\mathcal{M}_i$. Let $d_{i,m}$ be the duration of activity i when processed in mode m. Mode m is characterized by a resource demand profile $r_{i,m,t}$ which defines the resource demand on resource $k(i)$ in the processing periods $1, \ldots, d_{i,m}$. Due to technological constraints and management policies the resource demand profile has to take the following guidelines into account: i) Once processing of an activity is started it has to be continued until the end of the activity, i.e. no preemption is allowed. ii) The resource demand of activity i in each processing period has to be within an interval $[1, r_{k(i)}^{\max}]$ which depends on the resource $k(i)$ requested by activity i. I.e. $1 \leq r_{i,m,t} \leq r_{k(i)}^{\max}$ for $t = 1, \ldots, d_{i,m}$. $r_{k(i)}^{\max}$ is the maximum per period capacity requirement allowed for resource $k(i)$ where activity i has to be processed. The limitation to $r_{k(i)}^{\max}$ is due to the fact that splitting up the work content of an activity to be processed by more than $r_{k(i)}^{\max}$ laboratory units is not efficient because the overall setup time increases to much. iii) The resource demand has to be constant where an exception can hold for the last processing period.

These guidelines for the mode generation are similar to the ones of the discrete time/resource trade–off problem (DTRTP) in project networks where a single project has to be scheduled subject to a single renewable resource (cf. e.g. De Reyck et al (1998)). The difference is that for the DTRTP the modes have the same resource demand for each processing period whereas guideline iii) of the problem treated here prefers a constant resource demand but does not strictly require it.

In addition to real modes where work is performed by the use of resources, we will add a dummy mode for each activity i in its set of modes $\mathcal{M}_i$ where no resources are demanded and consequently no work is performed. The dummy mode will allow us to model the decision that a project is not selected. For the sake of brevity in notation we continue to use the same symbol to the denote the original set of modes plus the dummy mode; henceforth $\mathcal{M}_i := \mathcal{M}_i \cup \{0\}$ holds for each activity i.

Taking into account the guidelines stated above and denoting with m_i the number of non–dummy modes of activity i, we generate for each activity i the modes as follows: (a similar procedure is proposed for the DTRTP in De Reyck et al (1998)):

```
$d_{i,0} = 0$, $m_i = \min\{r_i, r_{k(i)}^{\max}\}$
For $m = 1$ to $m_i$:
    $d_{i,m} = \lceil \frac{r_i}{m} \rceil$
        $r_{i,m,t} = m$ for $t = 1, \ldots, d_{i,m} - 1$
        $r_{i,m,d_{i,m}} = r_i - m \cdot (d_{i,m} - 1)$
```

The first line sets the duration of the no–selection mode to zero and calculates the number of non dummy modes. Line three sets the duration of mode m to the number of periods needed if no more than m capacity units are used in each period. Line 4 sets for all but the last period of the duration the resource demand to m. Finally, line 5 sets the resource demand of the last period of the duration to the demand which has not been fulfilled in the preceding periods.

We will illustrate the mode generation with the following example. Let the work content of activity i be $r_i = 11$ on resource $k(i) = 1$ (chemistry). For chemistry, the maximum number of laboratory units which can work simultaneously in order to perform the activity is $r_1^{\max} = 4$. We obtain four modes plus the dummy mode 0 (cf. Table 13.2).

13.3.1.3 Time Horizon and Valuing of Work. The research committee of the lead optimization phase has to decide every quarter on the selection and scheduling of the projects with the objective to maximize the weighted work performed within the next 3 months (cf. Section 13.2.3). The point in time three months ahead is denoted with T_v while we denote with $T_{\max}$ the point in time until the last project will be finished. $T_{\max}$ is on average more than 2 years in the future. In order to obtain activity weights which link with the overall objective of value generation we proceed as follows. We first calculate the value of each

Table 13.2. Example of the Mode Generation

m	$d_{i,m}$	$(r_{i,m})$
0	0	
1	11	(1,1,1,1,1,1,1,1,1,1,1)
2	6	(2,2,2,2,2,1)
3	4	(3,3,3,2)
4	3	(4,4,3)

lead optimization project at the time it will be finished successfully. Next we determine activity weights by distributing the project value to its activities. Finally, we calculate the weight of each activity depending on its mode and start time. The weighted work accomplished until T_v will be maximized in the objective function. A formal description of the objective function will be given in Section 13.3.2. Let us now describe the three steps outlined above in detail.

Step 1: The net present value v_j^p of project j is calculated based on standard financial theory. We employ a decision tree approach which takes into account technological as well as market risks, future cash flows arising in all successor phases of the lead optimization phase, the option to terminate the project, and the time value of money (cf. for a similar approach Loch and Bode-Greuel (2001)).

Step 2: We distribute the value v_j^p of project j among its activities based on their work content. The weight of activity i is thus $w_i = v_{p(i)}^p \cdot r_i / r_{p(i)}^p$ where r_j^p is the work content of project j and $p(i)$ is the project where activity i is part of.

Step 3: For each activity i we calculate $w_{i,m,t}$ the weighted work accomplished until T_v if activity i is performed in mode m and started at time t.

$$w_{i,m,t} = w_i \cdot \frac{\sum\limits_{\tau=t}^{\min\{t+d_{i,m}-1, T_v-1\}} r_{i,m,\tau-t+1}}{r_i} \tag{13.1}$$

$w_{i,m,t}$ is equal w_i as long as activity i finishes not later than T_v, it is equal to a proportion of w_i if activity i is scheduled to start before T_v and finishes after T_v, and it is equal to 0 if activity i starts after T_v.

Table 13.3 gives a summary of the parameters introduced in this Section.

13.3.2 Integer Program

We employ binary decision variables $x_{i,m,t} = 1$, if activity i starts in mode m at time t, 0, otherwise, and the integer auxiliary decision variables $d_i(x)$,

Table 13.3. Sets and Parameters

Sets		Parameters	
$\mathcal{V}$	Set of activities	R_k	Capacity of resource k
$\mathcal{E}$	Set of precedence constraints	$r_k^{\max}$	maximum per period resource demand of a single activity on resource k
$k(i)$	Resource requested by activity i	r_j^p	work content of project j
$\mathcal{P}$	Set of projects	r_i	work content of activity i
$\mathcal{V}_j$	Set of activities in project j	$r_{i,m,t}$	capacity requirement of activity i in processing period t when processed in mode m
$\mathcal{M}_i$	Set of modes of activity i	$d_{i,m}$	duration of activity i in mode m
$\mathcal{R}$	Set of resources	$p(i)$	project to which activity i belongs
		v_j^p	value of project j
		w_i	weight of activity i
		$w_{i,m,t}$	weighted work obtained if activity i is performed in mode m and started at time t
		$\overline{d}_j$	deadline of project j
		T_v	time horizon of the objective function
		$T_{\max}$	time horizon of the constraints

the duration of activity i, and $S_i(x)$, the start time of activity i. The auxiliary variables are derived from the binary variables as follows: $d_i(x) = \sum_{m\in\mathcal{M}_i} d_{i,m} \sum_{t=0}^{T_{\max}} x_{i,m,t}$ and $S_i(x) = \sum_{m\in\mathcal{M}_i} \sum_{t=0}^{T_{\max}} t \cdot x_{i,m,t}$. Note that when activity i is processed in mode m and started at time t it will be processed in periods $t+1$ to $t+d_{i,m}$. We can now formulate the optimization problem as follows:

$$\text{Max} \quad \sum_{i\in\mathcal{V}} \sum_{m\in\mathcal{M}_i} \sum_{t=0}^{T_v} w_{i,m,t} \cdot x_{i,m,t} \tag{13.2}$$

subject to

$$S_l(x) - S_i(x) \geq d_i(x) \qquad \langle i,l\rangle \in \mathcal{E} \tag{13.3}$$

$$S_i(x) + d_i(x) \leq \overline{d}_j \qquad i \in \mathcal{V}_j \tag{13.4}$$

$$\sum_{\substack{i\in\mathcal{V}\\ k(i)=k}} \sum_{m\in\mathcal{M}_i} \sum_{\tau=\max\{0,t-d_{i,m}+1\}}^{t} r_{i,m,t-\tau+1}\cdot x_{i,m,\tau} \leq R_k \qquad t=0,\ldots,T_{\max};\ k\in\mathcal{R} \tag{13.5}$$

$$\sum_{m\in\mathcal{M}_i} \sum_{t=0}^{T_{\max}} x_{i,m,t} = 1 \qquad i\in\mathcal{V} \tag{13.6}$$

$$\sum_{t=0}^{T_{\max}} x_{i,0,t} = \sum_{t=0}^{T_{\max}} x_{l,0,t} \qquad i\in\mathcal{V};\ l\in\mathcal{V}_{p(i)} \tag{13.7}$$

$$x_{i,m,t}\in\{0,1\} \qquad i\in\mathcal{V};\ m\in\mathcal{M}_i;\ t=0,\ldots,T_{\max} \tag{13.8}$$

The objective function (13.2) maximizes the weighted work performed within the time window $[0, T_v]$.

The precedence constraints (13.3) depict the technological precedence relations between the activities of one project: Activity l can not start before each preceding activity i has been finished.

The deadline constraints (13.4) force all activities of project j to be completed not later than the deadline $\overline{d}_j$ so that the maximum project duration as given by management policy is kept.

The constraints (13.5) are the resource constraints. For each resource k the capacity demanded by the activities processed in period t has to be less than or equal the available capacity. The time interval for this constraint extends from 0 to $T_{\max}$ where $T_{\max}$ is the schedule horizon. That is the maximum number of periods needed to process all selected projects while not violating resource constraints. $T_{\max}$ can be determined as follows: $T_{\max} = \sum_{i\in\mathcal{V}} r_i$.

The constraints (13.6) are the activity execution constraints. Each activity i has to be performed in one of the modes $m \in \mathcal{M}_i$ which includes the dummy mode for not executing the project.

The constraints (13.7) are the mode identity constraints. The modeling concept of mode identity has been introduced by Drexl et al (1999) and Salewski et al (1997). For each project constraints (13.7) force all activities to be performed in mode 0 if this mode is selected for at least one activity of the project. Thereby we model the selection decision of the projects. The dummy mode 0 is associated with a work content of 0. Hence, in this mode no work will be performed on the activity, and due to constraints (13.7), on the project. Activities performed in mode 0 will have no contribution to the objective function according to formula (13.1).

Finally, constraints (13.8) define the decision variables.

The integer program (13.2) – (13.8) is a general formulation of the multi–mode resource constrained project scheduling problem with renewable resources (cf. Neumann et al (2003)) and hence a hard optimization problem.

Note that the number of variables in (13.2) – (13.8) can be reduced by introducing time windows for the start times of the activities (cf. Neumann et al (2003)). The latter are calculated for each project j by forward recursion from the earliest start time 0 and by backward recursion from the deadline $\overline{d}_j$, respectively. Also, the number of constraints (13.4) can be reduced by stating this constraint not for all activities of a project but only for the immediate predecessors of the dummy end activity.

13.4 Solution Methods

There are three types of decisions which have to be made. First, projects have to be selected, second, for each activity of a selected project the mode has to be determined, and third, the start time of the activity has to be set. We refer to the first decision as project selection and to the latter two as project scheduling. Project selection creates a project portfolio and project scheduling generates a multi–project schedule. In the following we will outline two different heuristic solution methods: sequential and concurrent project selection and scheduling. The sequential method starts with an empty project portfolio and iteratively selects one project which is then added to the portfolio. The concurrent method starts with the project portfolio comprising all available projects and iteratively deletes projects from the portfolio. Regardless of the particularities of the two approaches, both try to schedule activities as early as possible without explicitly taking into account the project deadlines. The latter are considered by a list–based shift method which will be described in Section 13.4.3.

13.4.1 Sequential Project Selection and Scheduling

The sequential method starts with an empty project portfolio and iteratively selects a project which is scheduled and added to the portfolio. The method stops after the last project has been scheduled and added to the portfolio. Note that in the case of a violation of the project deadline as given by constraint (13.4), the associated project is removed from the portfolio by assigning every activity of the project the mode 0.

13.4.1.1 Project Selection. For the project selection we use the following two straightforward criteria: i) We select the project with the highest project value (VAL), and ii) we select the project with the highest project value per work content (VPW). For the sake of being definitive we chose the the project with the smallest index of the start activity when there are ties.

13.4.1.2 Project Scheduling. In order to schedule a selected project we have to determine a mode and a start time for each activity of the project. Note that in the course of the sequential method (where some projects have already been scheduled) this problem is equivalent to the multi–mode project scheduling problem with time–varying resource availability (cf. Sprecher and Drexl (1998)). Hence we can employ a multi–mode adaptation of the serial schedule generation scheme (cf. Kolisch and Hartmann (1999)).

The serial schedule generation scheme (cf. Kolisch and Hartmann (1999)) iteratively selects an activity and schedules it at the earliest precedence– and resource–feasible time. Once all activities are selected and scheduled we have obtained an activity list which gives the order in which the activities have been selected. A close look at the problem treated here reveals that when scheduling the activities of a single project any order of activities will lead to the same schedule. This property is caused by the series–parallel structure of the research project where for the parallel activities within a cycle no two activities request the same resource. Hence, between the parallel activities we do not have so–called forbidden sets (cf. Bartusch et al (1988) and Brucker et al (1999)). Due to this property we can schedule the parallel activities of a cycle in any order. We choose the order given by the (consecutive) numbering of the activities.

For an activity which is chosen we have the following two rules for selecting a mode: i) We select the mode which will lead to the earliest finish time (EFT), or ii) we choose the mode with the shortest processing time (SPT). In case of ties we select the mode with the highest mode number. Note that due to the mode generation scheme presented in Section 13.3.1.2 this implies the mode with the shorter duration and thus higher resource demand per time.

13.4.2 Concurrent Project Selection and Scheduling

13.4.2.1 Project Selection. We start with a project portfolio which comprises all projects. For this portfolio we obtain a schedule and the associated objective function value by applying the schedule generation scheme described below. Thereafter we iteratively exclude one project from the portfolio and run the schedule generation scheme anew. We exclude the project with the smallest VAL– or alternatively the smallest VPW–figure of all projects still in the portfolio. In case of ties the project with the smallest index of the start activity is chosen.

13.4.2.2 Project Scheduling. The scheduling of the activities comprising the active portfolio is done with a hybrid schedule generation scheme. Based on the adjustment of the parameter θ the hybrid schedule generation scheme will function as a serial or a parallel schedule generation scheme and hence generate active or non–delay schedules (cf. Kolisch (1996)). The idea of a hybrid schedule generation scheme has originally been proposed by Storer et al (1992) for the job shop problem and has been adapted by Naphade et al (1997) for the resource–constrained project scheduling problem

Let us define $\mathcal{A}$ as the set of allowable activities. The allowable set contains all activities which have not yet been scheduled but can be scheduled since every predecessor has already been scheduled. For each activity $i \in \mathcal{A}$ we can calculate the earliest precedence– and resource–feasible start time S_i as it is done in the serial schedule generation scheme for selected activities (cf. Kolisch and Hartmann (1999)). Now we define $S^{\min}$ and $S^{\max}$ as the minimum and maximum precedence– and resource–feasible start time of the activities in the allowable set, i.e. $S^{\min} = \min\{S_i(x) \mid i \in \mathcal{A}\}$ and $S^{\max} = \max\{S_i(x) \mid i \in \mathcal{A}\}$. The decision set $\mathcal{D}$ is the set of all activities which can be selected for scheduling. Employing $\mathcal{A}$ it is defined as

$$\mathcal{D} = \{i \in \mathcal{A} \mid S^{\min} \leq S_i(x) \leq S^{\min} + \theta \cdot (S^{\max} - S^{\min})\} \qquad (13.9)$$

with $\theta \in [0, 1]$. For $\theta = 0$, activity $i \in \mathcal{A}$ needs a start time of $S_i(x) = S^{\min}$ in order to be in the decision set, whereas for $\theta = 1$, every activity i in the allowable set will be in the decision set since $S_i(x) \leq S^{\max}$ holds for all $i \in \mathcal{A}$. Obviously, we have for $\theta = 0$ the parallel scheduling scheme and for $\theta = 1$ the serial scheduling scheme, respectively.

We use the following two rules for the selection of an activity from the decision set: Choose the activity with the maximum VAL– or the maximum VPW–figure which is for an activity calculated as w_i and w_i/r_i, respectively. For the selection of the mode, we employ the same rules as for the sequential method.

13.4.3 Taking into Account Project Deadlines

Both, the sequential and the concurrent method have to take into account the project deadlines as given in (13.4). So far we have not addressed this issue from the methodical point of view. For the multi–mode RCPSP a method which takes into account maximum time lags which are a generalization of project deadlines has been presented by Heilmann (2001). Whenever the method detects the violation of a maximum time lag during the construction of a schedule, two steps are applied. The first step tries to enlarge the deadline by right shifting predecessors. It cannot be applied to our problem since the start of a maximum time lag has already been set in the lead generation phase and cannot be altered by the research committee of the lead optimization phase (cf. Section 13.2.3). The second step has a drawback since it tries to achieve the maximum time lag by squeezing together activities at the tail of the project which leads in our case to an uneven distribution of research output over time.

To obtain an even distribution of the research output we propose a list–based shift method. The backbone of the method is an activity list (cf. Kolisch and Hartmann (1999)) which lists all scheduled activities in the order they have been selected and assigned a start time. The method essentially works as follows: Whenever we detect a violation of a project deadline we employ the associated list by evenly shifting some of the scheduled activities of the project to the left. Afterwards we generate a new schedule by using the modified list. The left shifting of activities will lead to earlier start times and thus, hopefully, to a shorter makespan of the project under consideration. By considering the activities which will be left shifted in an even fashion we try to obtain an even distribution of the research output over time. We will now present the method more detailed:

Whenever an activity i is assigned a start time $S_i(x)$ it is checked whether the deadline of the associated project is still observed. In case of a violation of the project deadline we proceed as follows: As a result of the scheduling which has been done so far we have a partial list of length u where u is the number of activities which have already been scheduled. The list gives us the order in which activities have been set to their earliest (precedence– and resource–feasible) start times. For project $p(i)$ this list has caused a violation of the project deadline. We now try to shorten the duration of project $p(i)$ by assigning its already scheduled activities to lower labeled positions in the list. We search the list for a position $\hat{g}$ which fulfills the following two conditions: First, the activity $i(\hat{g})$ which is in position $\hat{g}$ belongs to project $p(i)$ under consideration, i.e. $i(\hat{g}) \in \mathcal{V}_{p(i)}$, and second, the number $c(\hat{g})$ of consecutive positions following position $\hat{g}$ where no activities of project $p(i)$ are placed is maximum. More formally, for all g with $i(g) \in \mathcal{V}_{p(i)}$ and $g < u$ the number $c(g)$ of consecutive positions following position g where no activities of project $p(i)$ are placed is given by

$$c(g) = \max\left(g, h \in \{g+1, \dots, u-1\} \mid \bigcup_{e=g+1}^{h} i(e) \bigcap \mathcal{V}_{p(i)} = \emptyset \right) - g \tag{13.10}$$

and the position $\hat{g}$ for which we have the maximum count as

$$\hat{g} = \max\left(g = 1, \dots, u-1 \mid c(g) = \max_{h=1,\dots,u-1} c(h) \right) \tag{13.11}$$

We now shift every activity of project $p(i)$ which is in a higher position than $\hat{g} + c(\hat{g})$ one position to the left. We illustrate the procedure referring to the example given in Figure 13.3 which consists of two projects. Project 1 comprises the activities 1 to 5 and project 2 comprises the grey shaded activities 6 to 9. When scheduling activity 9 according to the list displayed on the left side of Figure 13.3 a violation of the deadline of project 2 is recognized. Hence $u = 9$. Applying equation (13.10) for position $g = 2$ we obtain $c(g) = 3$ and applying equation (13.11) we obtain $\hat{g} = 2$. Performing left shifting, we obtain the new list displayed on the right side of the Figure 13.3.

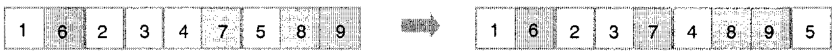


Figure 13.3. Example of the List–Based Shift Method

Cycling is prevented by forbidding the execution of exactly the same shift as just done for the next 10 iterations. The maximum number of times we apply the list–based shift method is set to 1000. When this number is hit during the course of the sequential or concurrent method, a project which exceeds (or will exceed) the project deadline is immediately discarded from the portfolio and the sequential or concurrent method is continued.

13.5 Experimental Evaluation

13.5.1 Test Instances

We have collected data from the pharmaceutical company. Because this essentially included only one instance, we opted to generate benchmark instances which are based on this instance but give a wider variety of problems. Furthermore, we wanted to study the impact of parameters which can be influenced by management or of which management wants to know how they impact profit. We therefore analyzed the real world instance and filtered out relevant parameters. We employed a full factorial test design where we differentiated into fixed, random, and systematically varied parameters. The test bed consists of 270 instances.

Fixed Parameters. The number of projects $| \mathcal{P} |$ is set to 100. The project values are normally distributed with a mean of 100. The standard deviation of the project values is systematically varied and will be detailed below. The horizon of the objective function is set to 12 (weeks). The number of renewable resources and their capacity is set according to Table 13.1.

Randomly drawn Parameters. The following parameters were randomly drawn from specified distribution functions. For each project the number of (remaining) cycles is drawn out of the uniformly distributed interval [2,15] where cycle times less than 10 depict running projects. The activities of one project within one cycle are generated as follows: First, the single start activity which is processed in the chemistry department and the first parallel activity which is processed in the eADMET department are generated. Afterwards, the activities which are processed in parallel in the subsections of the Disease Groups department have to be determined. Table 13.4 gives the probabilities that a project requests the Disease Groups sections Cardiovascular, Metabolic, or Rheumatic, respectively. As can be seen, in 98 % of the cases a project is processed in a single Disease Groups section. Only in 2 % of the cases one of the two sections Cardiovascular and Rheumatic or Metabolic and Rheumatic are requested, respectively. The ultimate request is not on the section but on the subsection level. Table 13.4 gives the conditional probabilities for the subsections Biochemistry, Molecular (Mol.) Biology, and Pharmacology for each section. Note that the probabilities and conditional probabilities of Table 13.4 determine the number of activities which are processed in parallel in the Disease Groups department. E.g. the probability that there are four parallel activities which are processed in eADMET and all three subsections of the section Cardiovascular as for project 1 in Figure 13.2 is $0.32 \cdot (0.37 \cdot 0.55 \cdot 0.79) = 0.055$.

Table 13.4. Probabilites of the Request for Sections and Conditional Probabilities of the Request for Subsections of the Disease Groups Department

Section	Probability	Subsection	Conditional Probability
Cardiovascular	0.32	Biochemistry	0.37
		Mol. Biology	0.55
		Pharmacology	0.79
Metabolic	0.38	Biochemistry	0.56
		Mol. Biology	0.44
		Pharmacology	0.82
Rheumatic	0.28	Biochemistry	0.67
		Mol. Biology	0.54
		Pharmacology	0.72
Cardiovascular and Rheumatic	0.01		
Metabolic and Rheumatic	0.01		

The resource demand for the activities is generated on account of the data given in Table 13.5. When generating the data, the interval given in the second column has been considered as follows: Whenever we have drawn a resource demand outside of the interval we have repeated the generation. Due to the cyclical structure of the project network (cf. Figure 13.2), each cycle of one project is alike.

Systematically varied Parameters. The following parameters were systematically varied: crash ratio, maximum duration factor, and standard deviation of project values.

Table 13.5. Distributions for the Generation of the Resource Demand

Departement	Distribution	λ	$E(r_i)$
Chemistry	uniform [2,10]		6
eADMET	exponential [1,5]	0.61	1.64
Disease Groups	exponential [1,7]	0.55	1.82

The **crash ratio** CR measures the ratio between the average duration of one cycle and the shortest possible cycle duration. The average duration for one cycle is 8 weeks and it is obtained if the project is processed in a "standard" way. The shortest cycle duration is obtained when all activities of the cycle are performed in their modes with shortest duration and the resource capacity is not scarce. CR has been set to the levels 1.1, 1.2, and 1.3. I.e. for CR=1.3 the cycle duration can be crashed to 6.1 weeks. Higher CR values have two effects: First, the cycle duration and thus the project makespan can be shortened which allows to create more weighted work within the time horizon of the objective function. Second, due to a larger number of modes, there is more scheduling flexibility which additionally gives the option to improve the objective function value.

The **maximum duration factor** MDF is the ratio of the average project duration if the project is processed with an average cycle duration and the maximum duration of the project as given by the management policy. MDF has been set to the levels 0.9, 0.8, and 0.7. For larger MDF–values the constraints (13.4) become harder and thus c.p. the objective function value should decrease.

SD is the **standard deviation of the project values** employing a normal distribution. SD has been set to levels of 5, 10, and 15. A higher standard deviation gives more variability of the project values and thus the opportunity to increase the weighted work by performing smart selection and scheduling decisions.

All random numbers were generated by employing the random number generator proposed by Matsumoto and Nishimura (1998). Employing a full factorial design with the three independent factors crash ratio CR, maximum duration factor MDF, and standard deviation SD and generating 10 replications for each combination of the factor levels we obtain $10 \times 3^3 = 270$ instances.

13.5.2 Results

13.5.2.1 Solution Methods and Bounds. First, we report on the performance obtained by the methods proposed in Section 13.4. We have solved all 270 test instances with each (variant of a) method. We always use random selection (RAN) as a benchmark. A good priority rule should produce considerably better results than random selection. The performance of the methods is measured by the average deviation from a lower bound (LB) and an upper bound (UB). The lower bound for each instance is the best solution obtained by all methods. The upper bound is calculated by solving a mixed integer program instead of the integer program (13.2) – (13.8). The mixed integer program is derived from the integer program by the following relaxations: First, we do not schedule activities but determine project intensities. The intensity of each project can vary within a range which is given by the shortest and the longest processing time of the project as defined by the parameters CR and MDF, respectively. A low (high) project intensity corresponds with a low (high) value in the objective function and a low (high) resource demand in the constraint. Second, for each resource a capacity constraint is not formulated for each period but for the whole planning horizon. The mixed integer program selects projects and sets intensities for selected projects so that the value is maximized.

Sequential Method. Table 13.6 gives the performance of the sequential method if performed as deterministic single pass (DET) and as biased random sampling (BRS) procedure with a sample size of 10 and 100, respectively (cf. Kolisch and Hartmann (1999)). For the project selection decision we have the ranking VPW $\succ$ VAL, RAN and for the mode selection we have the ranking EFT $\succ$ SPT $\succ$ RAN. The best results were obtained when applying the VPW–rule for the project selection and the EFT–rule for the mode selection. A sampling procedure with these two rules does not improve the performance of the method considerably.

Table 13.6. Results for the Sequential Method

		Selection of		Performance	
Method	# solutions	Project	Mode	LB	UB
DET	1	RAN	EFT	12.42	19.22
DET	1	RAN	SPT	12.57	19.52
DET	1	RAN	RAN	13.93	20.61
DET	1	VAL	EFT	11.54	18.40
DET	1	VAL	SPT	12.61	19.40
DET	1	VAL	RAN	14.20	20.85
DET	1	VPW	EFT	9.76	16.76
DET	1	VPW	SPT	11.82	18.66
DET	1	VPW	RAN	13.19	19.93
BRS	10	VPW	EFT	8.78	18.56
BRS	100	VPW	EFT	8.01	15.32

Concurrent Method. For all test instances, the concurrent method has been employed with 11 runs where the portfolio size has been decreased from 100 (which is the number of projects available) to 90. With this basic setting, we have first tested different values of θ in order to adjust the hybrid schedule generation scheme properly. Based on the results obtained with the Sequential Method, the project and activity selection has been set to VPW, and the mode selection has been performed with the EFT–rule. Table 13.7 gives the results of this study. The following can be observed: The parallel scheduling scheme ($\theta = 0$) clearly outperforms the serial scheduling scheme ($\theta = 1$). The next observation is that the hybrid schedule generation scheme with $\theta = 0.3$ produces better results than the parallel scheduling scheme. This result is in line with findings documented in the literature for the resource constrained project scheduling problem (RCPSP) where a single project has to be scheduled such that the makespan is minimized. Leon and Ramamoorthy (1995) as well as Naphade et al (1997) have found out that multi–pass methods such as the concurrent method produce the best results with $0 < \theta \leq 0.3$. Hence, we used the hybrid schedule generation scheme with $\theta = 0.3$ for the remainder of the study.

Table 13.7. Results of the Concurrent Method for Varying θ

	θ				
	0	0.3	0.6	0.9	1
LB	2.77	1.01	4.71	7.77	9.76
UB	10.33	8.71	12.12	14.93	16.76

Table 13.8 shows the performance of the concurrent method for the different project, activity, and mode selection rules proposed. The base method is as follows: $\theta = 0.3$, project and activity selection with VPW, and mode selection with EFT. For the project and the activity selection rule we can see that both rules, VAL and VPW outperform RAN and that the best method is clearly VPW. The same holds for the mode selection rules EFT and SPT if compared with RAN where EFT gives the best results.

13.5.2.2 Problem Parameters. We will now analyze the impact of the problem parameters MDF, CR, and SD on the performance of the best heuristic, i.e. the concurrent method with $\theta = 0.3$, EFT mode selection and VPW project and activity selection, respectively.

Maximum Duration Factor. Table 13.9 shows the impact of parameter MDF by reporting the average objective function value and the deviation from the average objective function value for each instance subset and for all in-

Table 13.8. Results of the Concurrent Method for Varying Project, Activity, and Mode Selection Rules

		Priority Rule		
Selection		VPW	VAL	RAN
Project	LB	1.01	1.99	4.00
	UB	8.71	9.62	11.47
Activity	LB	1.01	3.27	6.88
	UB	8.71	10.79	14.12
		Priority Rule		
Mode		EFT	SPT	RAN
	LB	1.01	2.33	7.13
	UB	8.71	9.87	14.34

stances. It can be seen that shorter maximum project durations do not affect the objective function. Hence, the management policy to prevent project delay by setting maximum project duration can be applied without sacrificing objective function value. But the shorter maximum project durations force the list procedure presented in Section 13.4.3 to be invoked more often which increases the computation time by a factor of 5. Two factors might cause the observed effects. First, the concurrent method is capable of selecting high value projects which are then scheduled in a more compact manner and second, the list–based shift method is capable to produce feasible schedules for tighter maximum project duration if more computer time is employed.

Table 13.9. Impact of the Maximum Duration Factor (MDF)

	MDF			
	0.7	0.8	0.9	
Avg. Objective Function Value	3002.08	3005.56	3000.90	3002.84
Deviation from avg. Value	−0.02	0.09	−0.06	

Crash Ratio. Table 13.10 gives the impact of the factor CR. It can be seen that with increasing crash ratio the objective function value raises as well. Hence, the conjecture made above is approved: Allowing shorter cycle times creates more weighted work by reducing the makespan of research projects and increasing scheduling flexibility. The volume for improvement is 6.68 %.

Table 13.10. Impact of the Crash Ratio (CR)

	CR			
	1.1	1.2	1.3	
Avg. Objective Function Value	2904.68	2998.13	3105.72	3002.84
Deviation from avg. Value	−3.26	−0.15	3.42	

Standard Deviation. Table 13.11 gives the impact of the standard deviation SD. It can be seen that a greater diversity of project values as measured by a higher standard deviation allows the concurrent method a better choice of high value projects. The volume for improvement is 2.38 %.

Table 13.11. Impact of the Standard Deviation (SD)

	SD			
	5	10	15	
Avg. Objective Function Value	2963.08	3010.69	3034.77	3002.84
Deviation from avg. Value	−1.32	0.26	1.06	

13.6 Summary

We have addressed the topic of selection and scheduling of pharmaceutical research projects within the lead optimization phase. The problem has been modeled as static multi–project multi–mode resource–constrained project scheduling problem. We have proposed two solution procedures: the sequential and the concurrent selection and scheduling method. Based on an experimental evaluation we show that the concurrent method produces superior results. From the management point of view we can point out that a higher variability of project values and an enlargement of the mode set towards modes with shorter activity duration and higher resource demand open the opportunity for generating more objective function value.

References

Bartusch, M., Möhring, R.H. and Radermacher, F.J. (1988). Scheduling project networks with resource constraints and time windows. *Annals of Operations Research* 16:201–240.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999). Resource–constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112(1):3–41.

Coffin, M.A. and Taylor, B.W. (1996). R&D project selection and scheduling with a filtered beam search approach. *IIE Transactions* 28:167–176.

De Reyck, B., Demeulemeester, E. and Herroelen, W. (1998). Local search methods for the discrete time/resource trade–off problem in project networks. *Naval Research Logistics* 45:553–578.

Drexl, A., Juretzka, J., Salewski, F. and Schirmer, A. (1999). New modelling concepts and their impact on resource–constrained project scheduling, in: *Project Scheduling — Recent Models, Algorithms and Applications* Węglarz, J., ed, , Kluwer Academic Publishers, Boston, pp. 413–432.

Fox, G.E., Baker, N.R. and Bryant, J.L. (1984). Economic models for R and D project selection in the presence of project interactions. *Management Science* 30(7):890–902.

Heidenberger, K. (1996). Dynamic project selection and funding under risk: A decision tree based MILP approach. *European Journal of Operational Research* 95:284–298.

Heilmann, R. (2001). Resource–constrained project scheduling: A heuristic for the multi–mode case. *OR Spektrum* 23:335–357.

Kolisch, R. (1996). Serial and parallel resource–constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research* 90:320–333.

Kolisch, R. and Hartmann, S. (1999). Heuristic algorithms for the resource–constrained project scheduling problem: Classification and computational analysis, in: *Project Scheduling — Recent Models, Algorithms and Applications*, J. Węglarz, ed, Kluwer Academic Publishers, Boston, pp. 147–178.

Kolisch, R., Meyer, K., Mohr, R., Schwindt, C. and Urmann, M. (2003). Ablaufplanung für die Leitstrukturoptimierung in der Pharmaforschung. *Zeitschrift für Betriebswirtschaft* 73(8):825–848.

Leon, V.J. and Ramamoorthy, B. (1995). Strength and adaptability of problem–space based neighborhoods for resource–constrained scheduling. *OR Spektrum* 17(2/3):173–182.

Loch, C.H. and Bode-Greuel, K. (2001). Evaluating growth options as sources of value for pharmaceutical resrearch projects. *R&D Management* 31(2):231–248.

Luh, P.B., Liu, F. and Moser, B. (1999). Scheduling of design projects with uncertain number of iterations. *European Journal of Operational Research* 113:575–592.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8(1):3–30.

Naphade, K.S., Wu, S.D. and Storer, R.H. (1997). Problem space search algorithms for resource–constrained project scheduling. *Annals of Operations Research* 70:307–326.

Neumann, K., Schwindt, C. and Zimmermann, J. (2003). *Project Scheduling with Time Windows and Scarce Resources*. Springer, 2. edition.

Patterson, J.H., Słowiński, R., Talbot, F.B. and Węglarz, J. (1990). Computational experience with a backtracking algorithm for solving a general class of precedence and resource–constrained scheduling problems. *European Journal of Operational Research* 49:68–79.

Salewski, F., Schirmer, A. and Drexl, A. (1997). Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. *European Journal of Operational Research* 102:88–110.

Schöffski, O., Fricke, F.-U., Guminski, W. and Hartmann, W., editors (2002). *Pharmabetriebslehre*, Springer.

Sprecher, A. and Drexl, A. (1998). Multi–mode resource–constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research* 107(2):431–450.

Storer, R.H., Wu, S.D. and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science* 38(10):1495–1509.

Talbot, F.B. (1982). Resource–constrained project scheduling with time–resource tradeoffs: The nonpreemptive case. *Management Science* 28(10):1197–1210.

Taylor, B.W., Moore, L.J. and Clayton, E.R. (1982). R&D project selection and manpower allocation with integer nonlinear goal programming. *Management Science* 28(10):1149–1158.

Venkatraman, R. and Venkatraman, S. (1995). R&D project selection and scheduling for organizations facing product obsolescence. *R&D Management* 25(1):57–70.

Viswanadham, N. and Narahari, Y. (2001). Queueing network modelling and lead time compression of pharmaceutical drug development. *International Journal of Production Research* 39(2):395–412.

Chapter 14

GRID MULTICRITERIA JOB SCHEDULING WITH RESOURCE RESERVATION AND PREDICTION MECHANISMS

Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, Jan Weglarz
Poznan Supercomputing and Networking Center, Poland
naber@man.poznan.pl

Abstract

Grids link together computers, data, sensors, large scale scientific instruments, visualization systems, networks and people. They can provide very large pools of computer resources, enable distributed collaborations and deliver increased efficiency and on-demand computing capabilities. The complexity of Grids on one hand and the requirements towards performance and capability on the other hand call for efficient resource management and scheduling mechanisms. Such mechanisms must take into account not only the hardware and software resources, user jobs and applications, but also policies of the resource owners. Policies usually describe cost models for the resource usage, security mechanisms, quality of service of resource provisioning etc. The problem of scheduling jobs in real Grid environments is very difficult. Due to lack of time characteristics of jobs, and difficulties in characterizing the overall system, traditional OR techniques usually fail or achieve very weak results. Usually, best effort scheduling is the best option. There are, however, some ways to deal with the problems described above.

The main goal of this paper it to present some practical issues of scheduling Grid jobs. Methods and techniques described in the paper are used in a Grid scheduling system, called GRMS (Grid Resource Management System) developed at Poznan Supercomputing and Networking Center. GRMS is widely used in many Grid infrastructures worldwide.

Keywords: Grid computing, Grid resource management and scheduling, multicriteria decision support.

14.1 Introduction

Grid computing can be defined as coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations. More simply, Grid computing typically involves using many resources (compute, data, I/O, instruments, etc.) to solve a single, large problem that could not be performed on any one resource. Often Grid computing requires the use of specialized middleware to mitigate the complexity of integrating of distributed resources within an Enterprise or as a public collaboration.

Generally, *Grid resource management and scheduling* is defined as the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible. Grid applications compete for resources that are very different in nature, including processors, data, scientific instruments, networks, and other services. Complicating this situation is the general lack of data available about the current system and the competing needs of users, resource owners, and administrators of the system.

Grids are becoming almost commonplace today, with many projects using them for production runs. The initial challenges of Grid computing–how to run a job, how to transfer large files, how to manage multiple user accounts on different systems–have been resolved to first order, so users and researchers can now address the issues that will allow more efficient use of the resources.

While Grids have become almost commonplace, the use of good Grid resource management tools is far from ubiquitous because of the many open issues of the field. Some of the issues include:

- **Multiple layers of schedulers.** Grid resource management involves many players and possibly several different layers of schedulers. At the highest level are Grid-level schedulers that may have a more general view of the resources but are very "far away" from the resources where the application will eventually run. At the lowest level is a local resource management system that manages a specific resource or set of resources. Other layers may be in between these, for example one to handle a set of resources specific to a project. At every level additional people and software must be considered.

- **Lack of control over resources.** Grid schedulers aren't local resource management systems; a Grid-level scheduler may not (usually does not) have ownership or control over the resources. Most of the time, jobs will be submitted from a higher-level Grid scheduler to a local set of resources with no more permissions than the user would have. This lack of control is one of the challenges that must be addressed.

- **Shared resources and variance.** Related to the lack of control is the lack of dedicated access to the resources. Most resources in a Grid environment are shared among many users and projects. Such sharing results in a high degree of variance and unpredictability in the capacity of the resources available for use. The heterogeneous nature of the resources involved also plays a role in varied capacity.

- **Conflicting performance goals.** Grid resources are used to improve the performance of an application. Often, however, resource owners and users have different performance goals: from optimizing the performance of a single application for a specified cost goal to getting the best system throughput or minimizing response time. In addition, most resources have local policies that must be taken into account. Indeed, the policy issue has gained increasing attention: How much of the scheduling process should be done by the system and how much by the user? What are the rules for each?

- **Missing time characteristics of jobs and tasks to be scheduled.** In Grids it is not possible to know most of time characteristics of jobs a priori. Time characteristics depend strongly on the performance and workload of a resource that is finally assigned to a job. The exact times are known only after the job is finished. Sometimes the users are able to give an estimate for their jobs. However, these estimates are very often far from the actual execution times. Time prediction methods might be also used to minimize the impact of this issue on a schedule quality. Another issue is a job ready time parameter. The job ready time depends on performance of the network and size of the job in terms of data that has to be moved from a local resource to a destination resource. The size of the data is also very often not know a priori. Very often, especially when we deal with jobs with precedence constraints, that size of the data to be moved from e.g. job *n* to job *n+1* is known after the job *n* is finished and all the files this job generates are written to a disk.

- **Lack of resource reservation mechanisms.** Another issue is lack of resource reservation mechanisms for most of the resources in Grids. Although there is a lot of work being done in this area there are still huge limitations and technical constraints when it comes to resource reservation mechanisms. We must say that most of the local resource management systems, those that are responsible for scheduling on *destination resource*, i.e. a resource which actually runs the job do not support resource reservation for remote Grid schedulers. Usually the only way to make a resource reservation is to make a phone call to a resource admin.

In this paper we will focus on the last two issues: missing time characteristics of jobs to be scheduled and lack of reservation mechanism in Grid systems. Generally, there are a few main motivations behind an adoption of resource reservation and prediction mechanisms in Grid resource management. First, additional knowledge about job start and completion times helps to improve an efficiency of scheduling in Grid since a Grid resource broker can make more appropriate decisions. These mechanisms are also essential for providing a Quality of Service (QoS) for end-users. This is important especially for certain classes of applications and scenarios, e.g. interactive applications or scheduling with deadlines, and if resource usage is charged because end-users want to know what they are charged for. In addition, use of knowledge about job start and completion times enables a Grid resource broker to schedule the whole set of jobs at the same time that should lead to a better overall allocation of resources.

Having the above as a main motivations behind this paper we will go further and will present the whole problem as a multicriteria choice problem, in which a scheduler, or resource broker, choses one of many schedules generated upfront, while scheduling some sets of jobs waiting in the system global queue.

The issues of Grid resource management and scheduling have been addressed only in part by the relevant literature in the field of Grid computing. The first book on Grid computing, *The Grid: Blueprint for a New Computing Infrastructure* by Foster and Kesselman, and its updated second edition, available in 2004, are a good starting point for any researcher new to the field. In addition, the book by Berman, Fox, and Hey entitled *Grid Computing: Making the Global Infrastructure a Reality*, presents a collection of leading papers in the area, including the "Anatomy of the Grid" and the "Physiology of the Grid", two papers that provide an overview of the shape, structure, and underlying functionality of Grid computing. The most complete set of approaches to resource management in Grids was presented in the book by J. Nabrzyski, J. Schopf and J. Weglarz entitled *Grid Resource Management: State of the Art and Future Trends, Kluwer Academic Publishers, November 2003 (Nabrzyski et al (2003)).* Research results on the topic of resource management in Grid environments are presented regularly in selected sessions of several conferences, including Supercomputing (SC), the IEEE Symposium on High-Performance Distributed Computing (HPDC), and the Job Scheduling Strategies for Parallel Processing workshop, as well as in the Global Grid Forum, a standards body for the field. The chapter is structured as follows: Section 2 gives an overview of resource reservation in Grid systems. Section 3 shows how prediction mechanisms can help with scheduling jobs in Grid environment. Next, in Section 4 we present a scenario of Grid job scheduling with predictions and resource reservations. We introduce a general model of multicriteria choice problem, which is one of the scheduling strategies in the GRMS (Grid Resource Management System) scheduling framework. GRMS itself is described in Section 5 and its practical usage example in Section 6. We conclude with the summary section in which we also sketch out our future research.

14.2 Resource Reservation in Grid Systems

Why is resource reservation in Grids so important? Let us give an example showing the main issues of resource reservation in Grids. A possible Grid scheduling scenario is presented in the Figure 1 below. The picture is not very much different from classical scheduling of machines. Of course none of time characteristics given in the picture are known a priori in Grids, as already explained above.

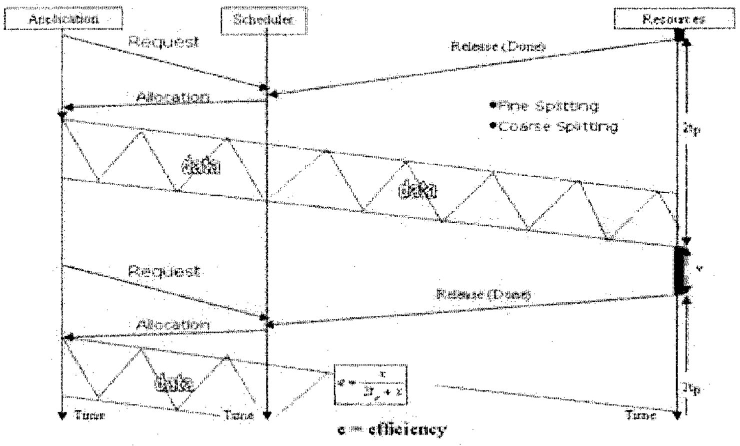


Figure 14.1. Most common Grid scenario: resource reservation is not supported by resources.

In the example a resource is allocated to a job for an unspecified amount of time, starting at the time the job-to-resource assignment is decided. As far as the information available at scheduler is concerned, the resource remains allocated to a job until the scheduler is informed about the job completion. The scheduler has to wait for a *release message* from the resource before it can allocate the resource to a new job. When a release message is received at the scheduler, an *allocation message* is sent to the application informing the user-side of where the available data for the execution of the job should be sent. The data are then transferred, which takes some additional communication time. During the transmission of the allocation message and during the transmission of the data to the resource, the resource remains (unnecessarily) inactive. When all the data are submitted the job begins execution on the resource. After the job is completed, the resource remains again (unnecessarily) inactive until the scheduler receives the release message so that it can then allocate it to another job. The figure shows actually the scheduling scenario when no resource reservation is applied.

We denote by $2t_p$ the average time that elapses between the time the resource sends the release message to the scheduler to inform it about the completion of a job until the time all the data required to execute the next job arrive at the resource. We also denote by x the average execution time of a job. It is then clear that Grid scheduling without resource reservation is performed the efficiency with which a resource is used is at most

$$e = \frac{x}{2t_p + x} \quad .$$

Note that the efficiency factor e may be considerably smaller than 1, and it also gets smaller as X decreases or as 2t$_p$ increases ($2t_p$ is at least as large as the roundtrip propagation communication delay). In order for the Grid to be useful for a number of different applications, we would like to be able to use fine grain computation (where x is small) and also be able to use remote resources (where $2t_p$ is large), both of which correspond to small values for the efficiency factor e.

This scenario shows also that some additional overhead must be taken into account when scheduling on the Grid. This overhead is caused by additional communication that must take place between the scheduler and the jobs that are to be run on potentially remote machines.

Let us now see how the efficiency factor e would look like with resource reservation supported by the remote resource. In the example we assume that a remote resource can be reserved in advance. It means we know when the resource will be fully available for our job so we can send all the data to be placed on the resource when the time to run a job comes.

The main idea behind advance/timed resource reservation is that the resources are reserved only for the time during which they are actually used for a job. In order to do so, the scheduler needs to reserve some execution time in the resources in advance. Of course we still do not know exactly how long will it take to run a particular job, but resource reservation maximizes the efficiency of the resources and the efficiency factor e can get very close to 1.

14.3 Scheduling Grid Jobs Using Prediction Information

Most of existing available resource management tools use general approaches such as load balancing (Shirazi et al (1995)), matchmaking (e.g. Condor Condor (www)), computational economy models (Abramson et al (2002)), or multi- criteria resource selection (Kurowski et al (2000b)). In practice, the evaluation and selection of resources is based on their characteristics such as load, CPU speed, number of jobs in the queue etc. However, these parameters can be related to the actual performance of applications, which may be not known a priori by end users. Users usually do not know what is the exact influence of these parameters on job start (e.g. local queue waiting) and execution times at

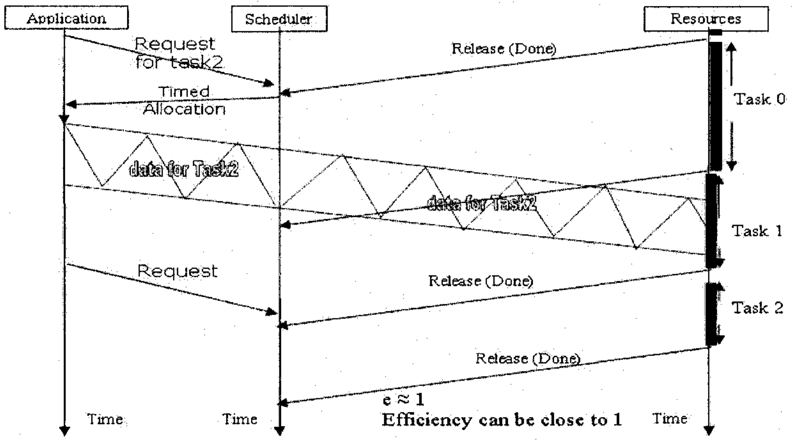


Figure 14.2. Resource reservations when advance and timed reservations are used.

different machines. Therefore, available estimations of job start and run times may significantly increase a quality of scheduling and, consequently, the overall performance. Nevertheless, due to incomplete and imprecise information, results of performance prediction methods may be accompanied by considerable errors (see Gibbons (1997), Smith et al (1999)). The more distributed, heterogeneous, and complex environment the bigger prediction errors may occur. Thus, these errors should be estimated and taken into consideration by a Grid scheduler while evaluating available resources or schedules. Our approach to estimating job start and run times has been presented in (Kurowski et al (2005)). This method takes into account estimated prediction errors to improve decisions of the Grid scheduler and to limit their negative influence on overall performance. In the method, the predicted job start- and run-times are generated by the Grid Prediction System (GPRES), developed in our collaboration with Wroclaw Supercomputing Center. Prediction techniques can be applied in a wide area of issues related to Grid computing: from the prediction of the resource performance in a near future to the prediction of the queue wait time (Smith et al (1999)). Most of these predictions are applied to resource selection and job scheduling. Prediction techniques can be classified into statistical, AI, and analytical. Statistical approached are based on applications that have been previously executed. These can be time series analysis (Dinda (2001), Wolski et al (1999)), categorization (Smith et al (1999), Downey (1997), Gibbons (1997), Kurowski et al (2000a)), and in particular correlation and regression have been used to find dependencies between job parameters. Analytical techniques construct models by hand (Schopf and Berman (1998)) or using automatic code instrumentation (Taylor et al (2001)). AI techniques use historical data and try to learn and classify the information in order to predict the future performance of resources or applications. AI techniques that can be used here are, for instance, classification (decision trees (Quinlan (1986)), neural networks (Rumelhart et al (1986))), clustering (k-means algorithm (Darken and Moody (1990))). Predicted times are used to predict resource information to guide scheduling decisions. Such scheduling process can be oriented to load balancing when executing in heterogeneous resources (Dail (2001), Figuiera and Bermann (2001)), or resource selection (Kurowski et al (2000b)) or used when multiple requests are provided (see Czajkowski et al (1997)). For instance, in Liu et al (2002) the 10-second ahead predicted CPU information is provided by NWS (Wolski (1997), Wolski et al (1999)). Many local scheduling policies, such as Least Work First or Backfilling, also use user provided or predicted execution time to make scheduling decisions (Lifka (1995), Feitelson and Mu'alem Weil (1998), Feitelson (www)).

In the approach presented in Kurowski et al (2005) we use the GPRES Expert System, which uses very simple template approach for predictions (Smith et al (1999)). The template is the set of job attributes, which are used to evaluate jobs similarity. Attributes for templates are generated from historical information after tests. Prediction process consists of several steps:

1 Initialize empty job category set C_z

2 For every template $T_i \in T$

- generate job category c_i
- add c_i to C_z

3 Initialize empty decision category set C_d

4 For every category c_i to C_z

- select categories from Knowledge Base corresponding to category c_i
- add categories to C_d

5 select best category from C_d

Where: d – decision attribute, T – template set, C – category set.

The method of selecting the best rule (category) can be set as a parameter to prediction module. Actually there are avaliable two methods in GPRES. The first one is based on number of condition attributes in rules. The most specific rule is chosen, i.e. the rule which has attributes of the job matched to the condition in the best way. The second strategy prefers a rule generated from the largest amount of history jobs. GPRES allows to mix these two methods in the way that if the first one gives still several rules the second is used. If both methods don?t give the final selection, the rules are combined and arythmetic mean of decision attribute is returned. Experiments presented in Kurowski et al (2005) proved that use of knowledge about estimated job completion times may significantly improve resource selection decisions made by Grid scheduler and, in this way, the performance of applications and the whole Grid system. Nevertheless, estimated job completion times may be insufficient for effective resource management decisions. Results of these decisions may be further improved by taking the advantage of information about possible uncertainty and inaccuracy of prediction. In the next section we will present the multicriteria Grid job scheduling model in which both, resource reservation and prediction mechanisms are used to improve scheduling performance.

14.4 Grid Job Scheduling with Predictions and Resource Reservations

Figure 3 presents a general Grid resource management scenario with resource reservation and time prediction mechanisms.

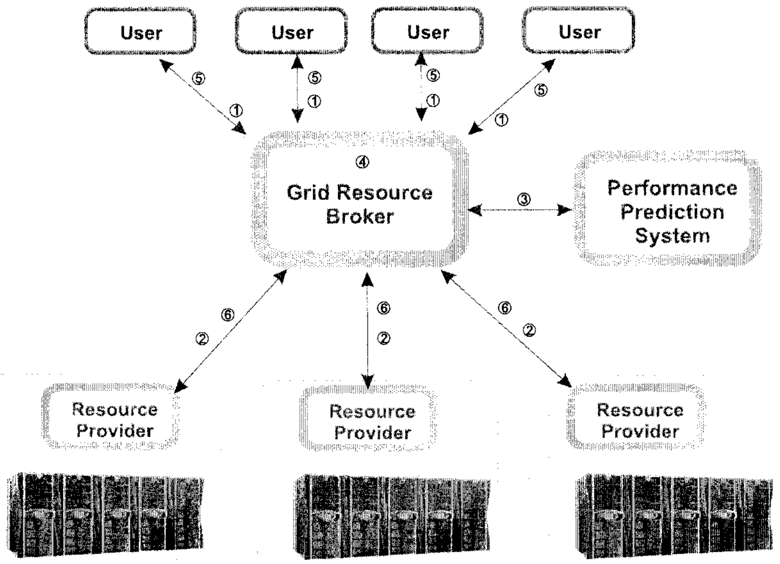


Figure 14.3. Grid resource management using resource reservation and performance prediction mechanisms

Resource broker after receiving resource requests from users (step 1) asks resource providers about their offers. Offers are returned in a form of lists of amounts of available resource units in various time slots (step 2). Providing an offer a resource provider agrees to initially reserve resources for a certain period. If a reservation is not confirmed before the end of this period the reservation is canceled. This approach guarantees that resources will not be reserved by other consumers. In the next step a performance prediction system provides knowledge about estimated job start and completion times (3). The prediction system calculate estimations based on historical information containing traces of previous job submissions. Following this a resource broker filters offers according to constraints defined by end-user, choses the best schedule (4) and returns this information to users or software acting on behalf of them (5). For users that have accepted a schedule given by the broker the reservations are confirmed with appropriate providers (6).

14.4.1 Model of the Scenario

In this section we define more formally a model adopted for the described scenario. End-users from a finite set $U = u_1, u_2, ..., u_{|U|}$ need to run their jobs $J = j_1, j_2, ..., j_{|J|}$ on resources belonging to resource providers from the set $RP = rp_1, rp_2, ..., rp_{|RP|}$. For each job resource requirements are defined. They are modeled as a set of hard constraints that must be met as explained in the previous sections. They consist of amounts of resource units RU^{req} that must be reserved for a given job (e.g. 3 CPUs, 1GB of disc space, etc.) and required resource attributes Q^{req} (e.g. CPU speed at least 1TFlops).

In this model we assume that a scheduler has knowledge about job start times. Thus, each resource provider must provide information about its offers in a form of lists containing available resource units in certain time slots in a given time period (t_0, t_f): $RT_i(t_o, t_f) = rt_{i1}, rt_{i2}, \ldots, rt_{ik}$, $k = |RT_i|$, $i = 1, \ldots, |RP|$, $rt_{ik} = (t_i^{start}, t_i^{end}, RU_{ik}, Q_{ik})$, where $RU_{ik} = (ru_{ik1}, ru_{ik2}, \ldots, ru_{ik|RU_{ik}|})$, and ru_{ikl} is an amount of the available resource unit l for resource provider i and time slot k (e.g. 100MB of free memory) that can be reserved for an end-user. Q_{jk} is a set of resource attributes as described in the previous sections (e.g. CPU speed, operating system, etc.).

In addition to knowledge of deterministic (guaranteed) job start times, information about estimated job execution and completion times is assumed to be available as explained above. Therefore, the Grid scheduler can take the advantage of the list of estimated job execution times, which can be calculated by the prediction system on the basis of resource attributes provided by each resource provider for a certain reservation: et_{ijk}^{exec} where $i = 1, ..., |J|, j = 1, ..., |RP|, k = 1, ..., |RT_i|$. Estimations are calculated on the basis of Q_{ik} - a specification of parameters describing a resource belonging to a given resource provider. Since job execution times are available and we assume that reserved jobs can start earlier if there is such a possibility, real job start times can also be estimated. These times, denoted as et_{ijk}^{start} where $i = 1, ..., |J|, j = 1, ..., |RP|, k = 1, ..., |RT_i|$, may be significantly shorter than the guaranteed ones. They can be provided either by a prediction system if this information cannot be taken from resource providers or by resource providers themselves (in the latter a broker or prediction system should estimate possible errors of predictions delivered by resource providers).

14.4.2 Multicriteria Choice Problem

The problem to be solved is to find the best time slot (resource providers' offer) for each job according to end-user's requirements. Each assignment of a job to a time slot $(j \rightarrow rt)$ is a candidate solution (also called action using a decision support terminology) and denoted as $a \in A$, where A is a set of all candidate solutions. Requirements consist of constraints that must be satisfied (hard constraints) and preferences concerning a choice of the best solution (soft constraints).

In the first step offers of resource providers must be filtered according to hard constraints defined by an end-user. This step can be performed by resource providers themselves since they retrieve information about job requirements from a resource broker in order to decide according to their local policies if there are any offers for a job. To this end two issues are cross-checked. For each job k and offer $rt_{ij}, i = 1, ..., |RP|, j = 1, ..., |RT_i|$ a resource broker (or provider) checks if requirements $Q_k^{req}, k = 1, ..., |J|$ concerning resource attributes Q_{ij} are met, i.e. whether $\forall_{q_{ijl} \in Q_{ij}} (q_{ijl} \propto q_{kl}^{req})$. $\propto$ denotes a relation between resource attribute (q_{ijl}) and a job requirement concerning this attribute (q_{kl}^{req}). This relation occurs if and only if q_{ijl} matches q_{kl}^{req}, e.g. is less, equal or greater than required values depending on particular attributes. In the second step it is checked if a sufficient amount of resource units can be reserved, i.e. whether $\forall_{ru_{ijm} \in RU_{ij}} (ru_{ijm} \geq ru_{km}^{req})$. It can be done again by a resource provider or a resource broker.

In addition to hard constraints (C) that must be satisfied, a Grid resource broker needs criteria (soft constraints) that define how the best resources should be selected. End-user can specify more than one soft constraint, e.g. time and cost. To handle such requests modeling and exploitation of multi-criteria users' preferences is needed.

Various models of preference modeling can be adopted in Grid resource management (Greco et al (1998)). In general we can distinguish two ways of preferences acquisition: (i) preferences are given explicitly by an end-user, e.g. in the form of criteria weights or criteria ranking, and (ii) end-users' preferences are discovered on the basis of their decisions (comparison of potential solutions). Which method should be used depends on two major aspects: first, whether users are familiar with basic concepts of decision support theory and

Table 14.1. Criteria

No	Symbol	Description
Cr_1	gt^{start}	Guaranteed job start time (according to an agreement concerning advance resource reservation)
Cr_2	$cost$	Total cost of reservation
Cr_3	$mean\ et^{exec}$	Estimated mean job execution time (based on job description and parameters of selected resource)
Cr_4	$stdev\ et^{exec}$	Estimated standard deviation of job execution time
Cr_5	$max\ et^{exec}$	Estimated maximal value of job execution time
Cr_6	$err\ et^{exec}$	Estimated prediction error of job execution time
Cr_7	$mean\ et^{start}$	Estimated mean job start time (based on estimated execution times of previously scheduled jobs)
Cr_8	$stdev\ et^{start}$	Estimated standard deviation of job start time
Cr_9	$max\ et^{start}$	Estimated maximal value of job start time
Cr_{10}	$err\ et^{start}$	Estimated prediction error of job start time

aware how to express preferences and, second, whether their preferences are relatively stable. If preferences change for each job submission, e.g. due to different application requirements or certain unpredictable aspects, an automated learning of users' preferences is very difficult. Then methods based on utility theory or lexicographic order of criteria can be applied.

In the presented model criteria considered in the decision support process are time and cost related. Nevertheless, in addition to the main criteria: guaranteed (reserved) job start time and cost, criteria that define estimated job execution time and start time are also taken into consideration. For both of these metrics the imprecision measures such as standard deviation, maximal value, and estimated prediction error are defined (as another criteria). The estimated execution time let differentiate the quality of available resources. The estimated start time can be significantly less then the guaranteed one since we assumed in the model that resource providers can shift jobs if previous ones have finished earlier. These values can be returned by a prediction system but also provided by a resource provider itself. The complete list of criteria used in the model is presented in Table 1. Of course, additional criteria can be easily added without reducing the generality of the model.

Although the set of criteria is quite big, in most cases only a subset of them is used. For instance, probably only one of prediction imprecision measures is relevant at the same time for an end-user. As mentioned above, different methods of preference modeling can be applied, however, here we present a procedure of resource selection using a utility function. For each pair: a job and time slot a utility function is calculated according to the formula:

$$F(j_k, rt_{ij}) = \sum_{l=0}^{|CR|} w_{kl} f_l(rt_{ij}), \tag{14.1}$$

$$f_l(rt_{ij}) = \frac{(Cr_{kl} - min_{kl})}{(max_{kl} - min_{kl})}, \tag{14.2}$$

where $i = 1, ..., |RP|, j = 1, ..., |RT_i|, k = 1, ..., |J|$, w_{kl} is a normalized weight of l^{th} criterion concerning job k, and $|CR|$ is a number of criteria (in this case $|CR| = 10$)

The max_k and min_k values are essential for appropriate scaling of criteria values. The function (1) is the simplest utility function consisting of one linear section. The advantage of this preference model is that this can be relatively easily and quickly defined by an end-user and calculation of utility function is immediate. Using this function a resource broker evaluates resources for one job only.

Based on the definitions, notations, and considerations described above the problem can be generally defined in the form of a multi-criteria decision support problem as follows:

$$min\{f_1(a), f_2(a), ..., f_{|CR|}(a)\}, \tag{14.3}$$

s.t.

$$\forall_{q_{ijl} \epsilon Q_{ij}} (q_{ijl} \propto q_l^{req}),$$
$$\text{where } q_{ijl} \epsilon Q_{ij},\ q_l^{req} \epsilon Q_k^{req},\ l = 1, ..., |Q|$$

$$\forall_{ru_{ijm} \epsilon RU_{ij}} (ru_m \geq ru_m^{req}),$$
$$\text{where } m = 1, ..., |RU|$$

$$a = j \rightarrow rt_{ij}, i = 1, ..., |RP|, j = 1, ..., |RT_i|$$

14.4.3 Scheduling of Job Sets

Knowledge about job completion times gives to a Grid resource broker a possibility to schedule more that one job at the same time. Doing this a Grid resource broker can try to optimize a whole schedule like in classical scheduling approaches. Otherwise, using online scheduling, an order of jobs in a queue may strongly influence a quality of a schedule. Additionally, if a broker schedules multiple jobs at once resource providers are asked to make preliminary reservations of their resources only once for all jobs. Note that in the presented model the main goal is to maximize a total satisfaction of end-users instead of fixed global criteria.

When multiple jobs are being scheduled a resource broker must get resource providers' offers not only for a single job. Therefore, resource provider should specify jobs that can be run in given time slots. Thus, for each time slot the following list must be provided: $JT(rt_{ij}) = \{j_1, j_2, ..., j_{|J|}\}$. Note that one time slot can be reserved for multiple jobs if there are enough resource units available in this slot.

A consequence of scheduling sets of jobs, which have come in a certain time interval, is a need for solutions that satisfy in the best possible way objectives of multiple end-users. Therefore, a total users' satisfaction must be evaluated. To this end, preferences of all end-users have to be aggregated into a measure that allows a resource broker to select the best schedule. A method of aggregation depends on an approach used for modeling user's preferences. If a utility function is used for criteria aggregation, an evaluation of the whole schedule is performed according to the following formula:

$$FT(J, RT) = \frac{1}{|J|}\sum_{k=0}^{|J|} F(j_k, rt_{ij}), \qquad (14.4)$$

where $i = 1, ..., |RP|, j = 1, ..., |RT_i|, k = 1, ..., |J|$ and rt_{ij} is a time slot chosen for job k.

In order to make this aggregation fair and reasonable min_{kl} and max_{kl} values in formula (2) must be specified carefully. They should define very bad and very good values of a criterion respectively from an end-user's perspective. Otherwise, if for example minimal and maximal values from a set of candidate solutions are taken as min_{kl} and max_{kl}, utility functions of different users are totally incomparable. In spite of the same values of these functions for two solutions evaluated by two different users, the real assessment of these solutions may significantly differ. Therefore, if min_{kl} and max_{kl} cannot be accurately defined other methods of aggregation should be used instead. For instance, if dependencies between solutions are given in a form of partial preorder of solutions the aggregation procedure based on Net Flow Score (Greco et al (1998)) can be applied.

The formulation of this problem differs from that defined for single-job scheduling described above. In this case a candidate solution is an assignment of jobs to time slots offered by resource providers. Generally multiple jobs can be assigned to one time slot. Additionally resource providers should return information which jobs can be assigned to a given time slot.

$$min\{f_1(a), f_2(a), ..., f_{|CR|}(a)\}, \qquad (14.5)$$

$s.t.$

$$\forall_{i,j}\forall_{k:(j_k \to rt_{ij})\epsilon a}(q_{ijl} \propto q_{kl}^{req}),$$

where $q_{ijl} \epsilon Q_{ij}, q_{kl}^{req} \epsilon Q_k^{req}, l = 1, ..., |Q|$

$$\forall_{i,j} (\sum_{k:(j_k \rightarrow rt_{ij}) \epsilon a} ru_{km}^{req}) \leq ru_{ijm},$$

where $ru_{ijm} \epsilon RU_{ij}, ru_{km}^{req} \epsilon RU_k^{req}, m = 1, ..., |RU|$

$$\forall_{(j_k \rightarrow rt_{ij}) \epsilon a} j_k \epsilon JT(rt_{ij})$$

$$a = \{j_1 \rightarrow rt_{ij}, j_2 \rightarrow rt_{ij}, ..., j_{|J|} \rightarrow rt_{ij}\}, i \epsilon \{1, ..., |RP|\}, \\ j \epsilon \{1, ..., |RT_i|\}, k \epsilon \{1, ..., |J|\}$$

The set a is a candidate solution (decision action). It consists of an ordered list of time slots assigned to every single job that belongs to the set J. The first constraint ensures that all time slots meet requirements of assigned jobs concerning resource attributes. The goal of the second constraint is to guarantee that sums of resource units that have to be allocated to assigned jobs do not exceed those offered by resource providers. As explained earlier Q_{ij} and RU_{ij} mean attributes of resources and amounts of resource units offered by resource providers respectively. Q_k^{req} and RU_k^{req} are corresponding job requirements concerning these values.

14.5 GRMS - An Example Grid Scheduling Framework

GRMS, (Kurowski et al (2001), Kurowski et al (2003), Kurowski et al (2004)), is an open source meta-scheduling system for large scale distributed computing infrastructures (Allen et al (2003)), developed at Poznan Supercomputing and Networking Center. Based on the dynamic resource selection, mapping and advanced Grid scheduling methodologies, it has been tailored to deal with job and resource management challenges in Grid environments, i.e. load-balancing among clusters, setting up execution environments before and after job execution, remote job submission and control, file staging, and more. GRMS was developed entirely in Java and thus can be installed on various kinds of operating systems and resources. GRMS is infrastructure independent and can be easily integrated with various Grid infrastructures, including all versions of Globus (Globus (www)), as well as enterprise Grids based on DRMAA-based infrastructures, including Condor, Sun N1GE (drmaa (www)) GRMS is able to take advantage of other middleware services, e.g. the Grid Authorization Service (GAS) or Replica Management Services, as well as to interoperate with infrastructure monitoring tools such as the GridLab's Mercury Monitoring System. One of the main assumptions for GRMS is to perform remote job control and management in the way that satisfies users' (job owners) requirements (Kurowski et al (2003)).

The main GRMS functionality includes: queuing submitted job, finding the best resources according to users' preferences, staging in/out files, submitting

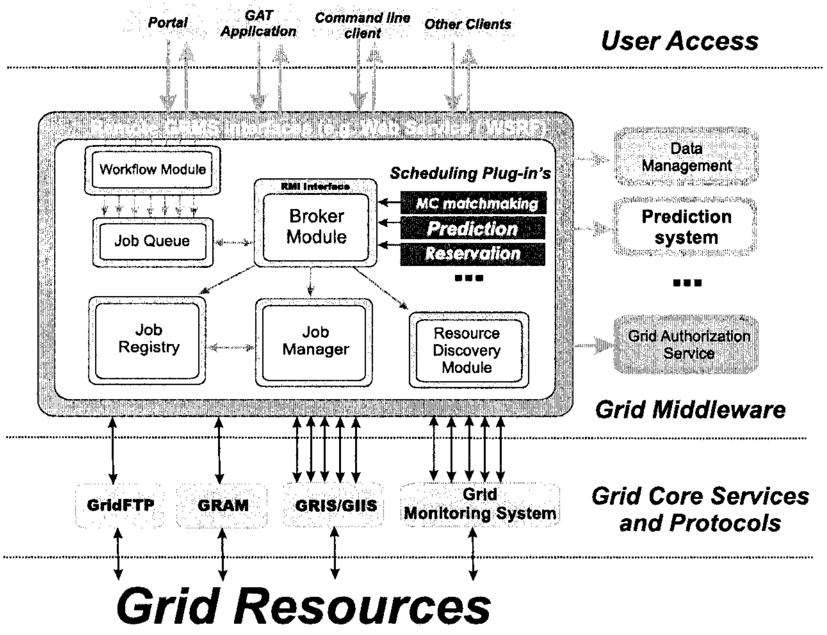


Figure 14.4. Detailed view of GRMS

job to, potentially remote computational resources, job migration, job canceling, logging, supporting workflow jobs.

Fig. 14.4 shows a more detailed view of GridLab GRMS with all its main modules and the Grid specific services, like *Replica Management*, *File Movement* and *Adaptive Components*.

As it is shown on fig. 14.4, GRMS consists of a set of various modules, including:

- **Broker Module**. The aim of the Broker Module is to control the whole process of resource and job management within the GRMS. This module steers a flow of requests to the GRMS and is also responsible for appropriate cooperation with other modules. Broker contains basic scheduling and policy strategies: matchmaking and multi-criteria matchmaking. The first strategy is a relatively simple but in fact very efficient approach for managing resources on which advanced reservation is not possible. The second strategy allows more flexible and accurate resource selections according to both users and administrator?s requirements and preferences. These two strategies can be easily modified and new scheduling and policy modules can be integrated as well.

- **Resource Discovery Module.** The Resource Discovery module monitors the status of distributed resources and therefore uses a flexible hierarchical access to both central and local information services. This module uses various techniques to discover and get an efficient access to up-to-date and accurate (both static and dynamic) information about jobs and resources. The goal of Resource Discovery it to deliver all information in a form and on time required by the Broker and its scheduling and policy strategies.

- **Job Manager Module.** The Job Manager module is responsible for monitoring of job status changes within the GRMS and then storing information in a database together with many additional parameters including resource requirements of jobs, user names, job IDs, submission times, pending times, execution times, jobmanagers to which jobs were submitted, history of migration if jobs have been migrated, etc. Due to the importance of historic information, especially in multi-site or large scale resource management systems, the GRMS provides also the interface for users and administrator to receive information about past GRMS actives. The tracking of historic resource utilization for all users results in the ability to modify job priorities, ensuring a balanced access, and optimizing administrator criteria (e.g. job throughput or turnaround time).

- **Job Queue** All the user requests come into the Job Queue and wait processing. Jobs in the queue can be scheduled one by one, in a simplest case (first in first out), or in collections, i.e. a number of jobs, or all the jobs, are scheduled in parallel. The Job Queue can be distributed across various Grid domains to allow multi-domain scheduling. In such case each domain has its own Broker instance. All the Brokers can transfer jobs between all the domains. The overall system state is controlled by inter-broker communication mechanisms.

- **Job Registry** is responsible for maintaining the database of all jobs submitted to GRMS and all information concerning those jobs.

External to GRMS is a prediction system. The idea here was to be able to communicate with external prediction services, or systems and so far GRMS has been integrated with GPRES Prediction Expert System mentioned in previous section.

14.5.1 Resource Reservation in GRMS

All information related to time requirements of interactive jobs is passed to the system during the job submission process as a part of job description. Every job to be submitted can have an optional section that defines in a formal way the time requirements for the job to be computed. This gives a user the possibility to build descriptions of advanced execution schedule in a simple and flexible way. The "execution time" section consists of three subsections

```
<grmsjob appid = "interactive_example">
  <simplejob>
    <executable type="single" count="1">
      <file name="exec-file" type="in">
        <url>file:///${HOME}/interactive_test/interactive_exec</url>
      </file>
    </executable>
    <executionTime>
      <timeSlot>
        <slotStart>10:30:00</slotStart>
        <slotEnd>13:15:00</slotEnd>
      </timeSlot>
      <execDuration>P0Y0M0DT2H20M0S</execDuration>
      <timePeriod>
        <periodStart>2005-05-01T00:00:00-00:00</periodStart>
        <periodDuration>P0Y0M10DT0H0M0S</periodDuration>
           <excluding>
             <weekDay>Saturday</weekDay>
             <weekDay>Sunday</weekDay>
           </excluding>
      </timePeriod>
    </executionTime>
  </simplejob>
</grmsjob>
```

Figure 14.5. Example job description with time reservations

defining following requirements: optional slot within the day when a job must be executed, mandatory execution time and optional time period when a job must be executed. The slot within the day is specified by start time of the slot and optionally end time of it or time duration. Specifying this time slot a user can require that the job must be started after some time and not later then some other time of a day. Mandatory information concerning duration of the job execution determines length of the period when a resource reservation is needed for a job. It is the only time characteristic that can be changed by the user after the job was submitted. If it doesn't violate the schedule it is possible to extend the execution time of the previously submitted and running job. Planing the job execution a user can specify time period when a job must be executed. The presented job description (see fig. 14.6) illustrates usage of this functionality specifying liberal requirements that the job should be executed within the first ten days of May except Saturdays and Sundays.

Based on dynamic resource selection and discovery, mapping and advanced scheduling methodology, combined with a feedback control architecture and support from other Grid middleware services, it deals with dynamic Grid environments and resource management challenges, such as load-balancing among clusters and various work-load systems, remote job control, file staging, advance reservation, scheduling jobs with precedence relations etc.

One of the main requirements for GRMS development was to perform remote job control and management in the way that would satisfy job and resource owners in terms of their preferences. Therefore, GRMS implements multicriteria procedures and optimization techniques to define and build various flexible resource management strategies.

14.5.2 Multicriteria Approach in GRMS

One of the most important modules of the broker is the multicriteria schedule evaluator (*MCEvaluator*). *MCEvaluator* implements various multicriteria models and tools that are applied to real Grid resource management and scheduling problems, including those with resource reservations and predictions. *MCEvaluator* is a framework that anyone can plugin into with new abstraction models. Main entities in GRMS include:

- Criteria (objectives, soft constraints)
- Hard constraints
- Solutions (e.g. resources, schedules etc. along with description parameters)
- Evaluator (decision point)

The framework contains also some multicriteria methods that GRMS uses for selection of best schedule. *MCEvaluator* provides support for a Grid scheduler to:

- Identify and select the best resource that a particular job will be computed on. In a workflow applications this process is repeated for every job being part of the workflow.
- Assign every available resource to predefined alternatives (classifying or sorting problem) or to order the alternative resource, as in ranking problem.
- Provide a performance tableau. In Grids, when the AI techniques are used it is often necessary to identify major distinguishing features of the resource or the whole schedule.

In order to solve the above mentioned tasks *MCEvaluator* can use many different methods, starting from outranking ELECTRE methods, through the utility functions aggregating the partial preferences on multiple criteria (MAUT, UTA, AHP) etc and finishing on the rules based methods. The features described above can be used for:

- Selecting the best resource to run a job on. This feature allows to choose best machine for a job, taking into account user preferences and host parameters, such as CPU load, total and free memory available for a job, number of CPUs, CPU speed, operating system etc.

- Select best queue at the remote resource to submit a job to. Potentially every resource is managed locally by the local resource management system (LRMS). LRMSs are usually based on queues that have different lengths and represent different policies of the resource owners. Selection of the best queue by *MCEvaluator* is based on the estimated job runtime and queue waiting time.
- Selection of the best job to be migrated. GRMS allows to use various dynamic strategies to manage jobs and resources, including job check-pointing and migration. By migrating a number of small jobs a Grid scheduler may allow to run bigger jobs on particular resource, which otherwise would have to wait longer in a queue. Cost of migration and resource characteristics are taken into account before decision is made.

Along with the *MCEvaluator* GRMS comes with a specialized multicriteria meta-language for expressing job descriptions and user preferences.

14.6 GRMS in Action

Knowledge acquired by the prediction techniques described in section 3 can be utilized in Grids, especially by resource brokers. Information concerning job run-times as well as a short-time future behavior of resources may be a significant factor in improving the scheduling decisions. A proposal of the multi-criteria scheduling broker that takes the advantage of history-based prediction information is presented in this section. For our experimental considerations we have chosen the Minimum Completion Time algorithm, which is one of the simplest algorithms that require estimated job completion times. It assigns each job from a Job queue to resources that provide the earliest completion time for a particular job.

Nevertheless, apart from predicted times, the knowledge about potential prediction errors is needed. The knowledge coming from a prediction system shouldn't be limited only to the mean times of previously executed jobs which fit to a template. Therefore, we also consider minimum and maximum values, standard deviation, and estimated error. These parameters should be taken into account during a selection of the most suitable resources. Mean time stays as the most important criterion, however, relative importance of all parameters depends on user preferences and/or characteristics of applications. For instance, certain applications (or user needs) may be very sensitive to delays that can be caused by incorrectly estimated start and/or run times. In such case a standard deviation, minimum and maximum values become considerably important. Therefore, a multicriteria resource selection is needed to accurately handle these dependencies. In our case we used the functional model for aggregation of preferences. That means that we used a utility function and all

```
For each job $J_i$ from a head of the queue
    For each resource $R_j$, at which this job can be
      executed
        Retrieve from the GPRES prediction system the
        estimated completion time of job $C_{J_i,R_j}$
        Assign job $J_i$ to resource $R_{best}$ so that

            $C_{J_i,R_{best}} = \min_{R_j} \left( C_{J_i,R_j} \right)$
```

Figure 14.6. Algorithm MCT (Abramson et al (2002))

$$F_{J_i,R_j} = \frac{1}{\sum_{k=1}^{n} w_k} \sum_{k=1}^{n} w_k * c_k \tag{14.6}$$

resources were ranked based on values of utility function. In detail, criteria are aggregated for job J_i and resource R_j by the weighted sum given according to formula (6).

where the set of criteria C (n=4) consists of the following metrics:
C_1 – mean completion time
C_2 – standard deviation of completion time
C_3 – difference between maximum and minimum values of completion time
C_4 – estimated error of previous predictions
and weights w_k that define the importance of the corresponding criteria. This method can be considered as a modification of the MCT algorithm to a multi-criteria version. In this way possible errors and inaccuracy of estimations are taken into consideration in MCT. Instead of selection of a resource, at which a job completes earliest, the algorithm chooses resources characterized by the best values of the utility function F_{J_i,R_j}. As described above the function is calculated taking as an input values of four criteria: $time_{J_i,R_j}$, err_{J_i,R_j}, $stdev_{J_i,R_j}$, max_{J_i,R_j}, -min_{J_i,R_j}.

These two algorithms have been implemented in GRMS using its multi-criteria selection framework of *MCEvaluator.*

```
For each job $J_i$ from a head of the queue
    For each resource $R_j$, at which this job can be
        executed
        - Retrieve from the GPRES prediction system the
          estimated completion time of job $C_{J_i,R_j}$ and
          $err_{J_i,R_j}$, $stdev_{J_i,R_j}$, $max_{J_i,R_j}$, $min_{J_i,R_j}$.
        - Calculate the utility function $F_{J_i,R_j}$
    Assign job $J_i$ to resource $R_{best}$ so that
```

$$F_{J_i,R_{best}} = \max_{R_j} \left(F_{J_i,R_j}\right)$$

Figure 14.7. Multicriteria MCT algorithm

14.6.1 Experiment

The system where the workload trace file was obtained from was a IBM SP2 System from Barcelona Supercomputing Center. The system, named Kadesh.cepba.upc.edu, was used with two different configurations: the IBM RS-6000 SP with 8*16 Nighthawk Power3 @375Mhz with 64 Gb RAM, and the IBM P630 9*4 p630 Power4 @1Ghz with 18 Gb RAM. A total of 336Gflops and 1.8TB of Hard Disk are available. All nodes are connected through an SP Switch2 operating at 500MB/sec. The operating system that they are running is an AIX 5.1 with the queue system Load Leveler. The workload was obtained from Load Leveler history files that contained around three years of job executions (178.183 jobs). Through the Load Leveler API, we converted the workload history files that were originally in a binary format. Analyzing the trace file we can see that total time for parallel jobs is approximately and order of magnitude bigger than the total time for sequential jobs, what means that in median they are consuming around 10 times more of CPU time. For both kind of jobs the dispersion of all the variables is considerable big, however in parallel jobs is also around an order of magnitude bigger. Parallel jobs are using around 72 times more memory than the sequential applications. In general these variables have significant amount of variability what may result in difficulties with predictions. In general users are not working with a big set of applications. In median, users submitted 9 different applications, and, also in median, they executed each application around 8 times. However, from the 98 users 22 of them had submitted in mean same applications more than 30 times. Taking into account only these users, the presented median increases until 56.11 observations for user and application. Similar conclusions can be applied with user groups. Although same groups in general are submitting in median 22 different applications, they are still submitting few than 7 times the same application. However, there are some groups that are submitting in median more times same applications, from 22 groups, there are 6 groups that are submitting in median more than 42.2 times same applications.

We performed two major experiments. First, we compared results obtained by the MCT algorithm with a common approach based on the matchmaking technique (job was submitted to the first resource that met user?s requirements). In the second experiment, we studied improvement of results of the prediction-based resource evaluation after application of knowledge about possible prediction errors. For both experiments the following metrics were compared: mean, worst, and best job completion time. The worst and best job completion values were calculated in the following way. First, for each application the worst/best job completion times have been found. An average of these values was taken as the worst and best value for comparison. 5000 jobs from the workload were used to acquire knowledge by GPRES. Then 100 jobs from the workload were scheduled to Job queue ofr GRMS. The results of the comparison are presented in figure below. In general, it shows noticeable improvement of mean job completion times when the performance prediction method was used. The least enhancement was obtained for the best job completion times. The multi-criteria MCT algorithm turned out to be the most useful for improvement the worst completion times.

14.7 Conclusions

In this paper we elaborated on Grid job scheduling using Grid schedulers with resource reservation and prediction mechanisms. We also proposed the multi-criteria resource evaluation methods based on knowledge of job start- and run-times obtained from the prediction system. As a prediction system the GPRES tool was used. We exploited the method of multi-criteria evaluation of resources from GRMS. Resource reservation mechanisms were also used to make sure that resources are available at the moment of job staging. We presented how diverse end-users' requirements and preferences concerning time and cost can be modeled using multi-criteria decision support techniques. Thanks to this approach end-users can express both hard constraints that must be satisfied and soft constraints that help a resource broker to find the best offers of resource providers. Furthermore, we showed how preferences of multiple end-users can be aggregated in order to find a compromise schedule. The hypotheses assumed in the paper have been verified. Exploitation of the knowledge about performance prediction allowed a resource broker to make more

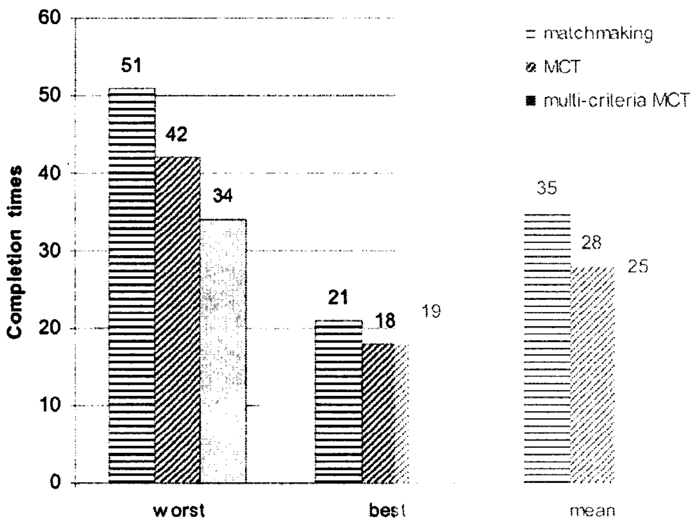


Figure 14.8. Comparison of job completion times for matchmaking, MCT, and multi-criteria MCT algorithms

efficient decisions. This was visible especially for mean values of job completion times. Exploitation of knowledge about possible prediction errors brought another improvement of results. As we had supposed it improved mainly the worst job completion times. Thus, taking the advantage of knowledge about prediction errors we can limit number of job completion times that are significantly worse than estimated values. Moreover, we can tune the system by setting appropriate criteria weights depending on how reliable results we need and how sensitive to delays the application are. For instance, certain users may accept 'risky' resources (i.e. only the mean job completion time is important for them) while others may expect certain reliability (i.e. low ratio of strongly delayed jobs). The preliminary results show that using prediction information and resource reservation can bring significant results, while scheduling jobs in Grid environments. Of course there are many limitations to apply the approach in open Grid systems, but, for many users and jobs, which run frequently in particular infrastructure the results may be impressive. In general, use of resource reservation and performance prediction mechanisms in Grids may help to improve performance by means of better Grid resource broker decisions (based on more accurate knowledge) and possibility of scheduling multiple jobs at once. Moreover, in certain scenarios in which QoS must be provided this approach is indispensable.

Nevertheless, there are some drawbacks and problems that must be taken into account when these mechanisms are used. First of all, an extensive use of resource reservation can deteriorate the overall job throughput. This unfavorable influence can be limited by use of accurate job execution time predictions (along with estimated imprecision) and resource providers policies that allow starting jobs earlier than their reserved start time. Convenient ratio of numbers of jobs with and without reservations is also a major factor that influences performance. The exact dependencies between these issues are a subject of further research. Another important issue is a need of additional steps to obtain offers of resource providers and estimations from a prediction system. These steps can also increase response time. Additionally, since we cannot assume a control of resource broker (and in consequence a prediction system) over local resources the prediction is more difficult due to limited information about resources. To solve this problem very advanced Grid monitoring systems need to be introduced.

Acknowledgments

The authors of this paper would like to thank Agnieszka Kwiecien, Maciej Dyczkowski and Marcin Wojtkiewicz from Wroclaw Networking and Supercomputing Center for integrating their GPRES systems with GRMS and making it available for the tests. We also wanted to thank Jesus Labarta from Barcelona Supercomputing Center for giving access to historical workloads of their machines.

References

Abramson, D., Buyya, R. and Giddy, J. (2002). A computational economy for Grid computing and its implementation in the Nimrod-G resource broker, *Future Generation Computer Systems*, 18(8):1061–1074.

Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules, in: *Proceedings of the Twentieth Intl. Conference on Very Large Databases*, Morgan Kaufmann, pp. 487–499.

Allen, G., Davis, K., Dolkas, K.N., Doulamis, N.D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Radke, T., Russell, M., Seidel, E., Shalf, J. and Taylor, I. (2003). Enabling Applications on the Grid - A GridLab Overview, *International Journal of High Performance Computing Applications*, 17(4):449–466.

Bode, B., Kendall, D.M. and Lei, Z. (2000). The Portable Batch Scheduler and the Maui scheduler on Linux clusters, in: *Proceedings of 4th Annual Linux Showcase and Conference*, October 2000.

Černy, V. (1985). Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm, *Journal of Optimization Theory and Applications*, 45:41–51.

Cheung, L.S. (2001). A Fuzzy Approach to Load Balancing in a Distributed Object Computing Network, in: *Proceedings of the First IEEE International Symposium of Cluster Computing and the Grid (CCGrid'01)*, pp. 694–699.

Condor Group, *Condor project*, http://www.cs.wisc.edu/condor.

Czajkowski, K., Foster, I., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. (1997). A resource management architecture for metacomputing systems, *JSSPP Whorskshop, Lecture Notes on Computer Science*, 1459:62–68.

Dail, H. (2001). A Modular Framework for Adaptive Scheduling in Grid Application Development Environments, Technical report CS2002-0698, Computer Science Department, University of California, San Diego.

Darken, C. and Moody, J. (1990). Fast Adaptive k-means clustering: Some empirical results, in: *Proceedings of the International Joint Conference on Neural Networks*, vol. II, IEEE Neural Networks Council, pp. 233–238.

Dinda, P. (2001). Online prediction of the running time of tasks, in: *Proceedings of 10th IEEE Symp. on High Performance Distributed Computing*, pp. 336–337.

Downey, A. (1997). Predicting Queue Times on Space-Sharing Parallel Computers, in: *11th International Parallel Processing Symposium*, pp. 209–218.

Global Grid Forum DRMAA WG, DRMAA Web Site, http://www.drmaa.org.

European DataGrid Project, http://www.eu-datagrid.org.

El-Ghazawi, T., Gaj, K., Alexandridis, N., Vroman, F., Nguyen, N., Radzikowski, J.R., Samipagdi, P. and Suboh, S.A. (2004). A performance study of job management systems, *Concurrency and Computation: Practice and Experience*, 16(13):1229–1246.

Feitelson, D.G. and Mu'alem Weil, A. (1998). Utilization and predictability in sche-duling the IBM SP2 with backfilling, *Proceedings of 12th International Parallel Processing Symp.*, Orlando, pp. 542–546.

Feitelson, D.G., Parallel Workload Archive, http://www.cs.huji.ac.il/labs/parallel/work-load.

Figuiera, S.M. and Bermann, F. (2001). Mapping Parallel Applications to Distributed Heterogeneous Systems, Technical report CS2002-0698, Computer Science Department, University of California, San Diego.

Foster, I. and Kesselman, C. (1998). The Globus Project: A Status Report, in: *Proceedings of the Seventh Heterogeneous Computing Workshop*, pp. 4–18.

Foster, I. and Kesselman, C. (editors) (1999). *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffmann, San Francisco, California.

Foster, I. and Kesselman, C. (1999). Computational Grids, in: *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds, Morgan Kaufmann, San Francisco, California, pp. 15–52.

Gibbons, R. (1997). A Historical Application Profiler for Use by Parallel Schedulers, *Lecture Notes on Computer Science*, 1297:58–75.

Globus Team, Globus Project, http://www.globus.org.

Glover, F. (1989). Tabu Search - part 1, *ORSA Journal of Computing*, 1(3):190–206.

Glover, F. (1990). Tabu Search - part 2, *ORSA Journal of Computing*, 2:4–32.

Glover, F. (1986). Future Path for Integer Programming and Links to Artificial Intelligence, *Computers & Operations Research*, 13:533–549.

Goldberg, D.E., (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading.

Greco, S., Matarazzo, B., Slowinski, R. and Tsoukias, A. (1998). Exploitation of a rough approximation of the outranking relation in multicriteria choice and ranking, in: *Trends in Multi-Criteria Decision Making*, T.J Stewart and R.C van der Honert, eds, Springer Verlag, Berlin, pp. 45–60.

Greco, S., Matarazzo, S. and Slowinski, R. (2001). Rough sets theory for multicriteria decision analysis, *European Journal of Operational Research*, 129(1):1–47.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

Ishibushi, H. and Murata, T. (1998). A Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling, *IEEE Transactions on Systems, Man and Cybernetics*, 28(3):392–403.

Jackson, D.B., Maui Admin Guide,
http://supercluster.org/maui/docs/mauiadmin.html.

Jaszkiewicz, A. (1998). Genetic local search for multiple objective combinatorial optimisation, Technical Report RA014 /98, Institute of Computing Science, Poznan University of Technology.

Kirkpatrick, S., Gelatt, C.D., Jr and Vecchi, M.P. (1983)., Optimization by Simulated Annealing, *Science*, 230:671–680.

Knowles, J.D. and Corne, D.W. (2000). A Comparison of Diverse Approaches to Memetic Multiobjective Combinatorial Optimization, in: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Workshop On Memetic Algorithms*, pp. 103–108.

Knowles, J.D. and Corne, D.W. (2000). M-PAES: A Memetic Algorithm for Multiobjective Optimization, in: *Proceedings of the 2000 Congress on Evolutionary Computation*, pp. 325–332.

Kurowski, K., Nabrzyski, J. and Pukacki, J. (2000). Multicriteria Resource Management Architecture for Grid, in: *Proceedings of the 4th Globus Retreat*, Pittsburgh, PA, July 2000.

Kurowski, K., Nabrzyski, J. and Pukacki, J. (2000). Predicting Job Execution Times in the Grid, in: *Proceedings of the 1st SGI 2000 International User Conference*, Kraków, pp. 272–282.

Kurowski, K., Nabrzyski, J. and Pukacki, J. (2001). User preference driven multiobjective resource management in Grid environments, in: *Proceedings of the First IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, pp. 114-121.

Kurowski, K., Nabrzyski, J., Oleksiak, A. and Węglarz, J. (2003). Multicriteria Aspects of Grid Resource Management, in: *Grid Resource Management*, J. Nabrzyski, J. Schopf, and J. Węglarz, eds, Kluwer Academic Publishers, Boston/Dordrecht/London, pp. 271–294.

Kurowski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A. and Pukacki, J. (2004). Improving Grid Level Throughput Using Job Migration and Rescheduling Techniques in GRMS, *Scientific Programming*, 12:(4)263-273.

Kurowski, K., Oleksiak, A., Nabrzyski, J., Guim, F., Corbalan, J., Labarta, J., Kwiecien, A., Wojtkiewicz, M. and Dyczkowski, M. (2005). Multicriteria Grid Resource Management using Performance Prediction Techniques, in: *Proceedings of the 2nd CoreGrid Workshop*, Springer Verlag (to appear).

Langley, P., Iba, W. and Thompson, K. (1992). in: *An Analysis of Bayesian Classifiers, Proceedings of AAAI-92*, pp. 223-228.

Lifka, D. (1995). The ANL/IBM SP scheduling system, in: *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson and L. Rudolph, eds, Springer-Verlag, Lecture Notes of Computer Science, 949:295–303.

Liu, C., Yang, L., Foster, I. and Angulo, D. (2002). Design and evaluation of a resource selection framework for Grid applications, in: *Proceedings if the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, pp. 63–72.

Nabrzyski, J., Schopf, J. and Węglarz, J., editors, (2003). *Grid Resource Management - State of the Art and Future Trends*, Kluwer Academic Publishers.

Nabrzyski, J. (2000). User Preference Driven Expert System for Solving Multiobjective Project Scheduling Problems, Ph.D. Thesis, Poznan University of Technology.

Pawlak, Z. (1982). Rough Sets, *International Journal of Information & Computer Sciences*, 11(5):341–356.

Platform Computing Technical Docs, http://www.platform.com/services/support /docs/LSFDoc51.asp.

Quinlan, J.R. (1986), Induction of Decision Trees, *Machine Learning*, 1:81–106.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning Representations by Back Propagating Errors, *Nature*, 323:533–536.

Sandholm, T.W. (1999). Distributed Rational Decision Making, in: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, ed, MIT Press, pp. 201–258.

Schopf, J. and Berman, F. (1998). Performance prediction in production environments, in: *Proceedings of IPPS/SPDP*, pp. 647–653.

Shirazi, B.A., Husson, A.R. and Kavi, K.M. (1995). *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press.

Smith, W., Taylor, V. and Foster, I. (1999), Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance, *Proceedings of the IPPS/SPDP '99 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 202–219.

Taylor V., Wu, X., Geisler, J., Li, X., Lan, Z., Hereld, M., Judson, R. and Stevens, R. (2001). Prophesy: Automating the modeling process, in: *Proceedings Of the Third International Workshop on Active Middleware Services.*

Veridian Inc. PBS: The Portable Batch System. http://www.openpbs.org/

Vazhkudai, S. and Schopf, J. (2003). Using Regression Techniques to Predict Large Data Transfers, *Journal of High Performance Computing Applications - Special Issue on Grid Computing: Infrastructure and Application*, 17: 249-268.

Węglarz, J., editor (1999). *Project Scheduling - Recent Models, Algorithms and Applications*, Kluwer Academic Publishers.

Wolski, R., Spring, N. and Hayes, J. (1999). The Network Weather Service: a distributed resource performance forecasting service for metacomputing, *Future Generation Computer Systems*, 15(5–6):757–768.

Wolski, R. (1997). Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service, *Cluster Computing*, 1(1):119-132.

Zadeh, L.A. (1965), Fuzzy Sets, *Information and Control*, 8(3):338–353.

Chapter 15

RESOURCE–CONSTRAINED PROJECT SCHEDULING WITH TIME WINDOWS

Recent developments and new applications

Klaus Neumann[1], Christoph Schwindt[2], Jürgen Zimmermann[2]
[1] *University of Karlsruhe, Germany;* [2] *Clausthal University of Technology, Germany;*
neumann@wior.uni-karlsruhe.de; {christoph.schwindt,juergen.zimmermann}@tu-clausthal.de

Abstract Recent results on resource–constrained project scheduling with time windows are reviewed. General temporal constraints (resulting from minimum and maximum time lags between project activities), several different types of scarce resources, and a large variety of time–based, financial, and resource–based objective functions are considered. Emphasis is placed on an order–based structural analysis of the feasible region of project scheduling problems and a classification and discussion of objective functions important to practice, which can be exploited for constructing efficient solution procedures. After those structural issues, methods for solving time–constrained project scheduling problems are proposed. Next, the resolution of conflicts for renewable, allocatable, synchronizing, changeover, and cumulative resources and thus the solving of corresponding resource–constrained project scheduling problems are studied. Finally, new applications of resource–constrained project scheduling are presented: factory pick–up of new cars and batch scheduling in process industries.

Keywords: Deterministic project scheduling, regular and nonregular objective functions, types of scarce resources, exact solution methods, customer–oriented factory pick–up, batch scheduling.

15.1 Introduction

Since the appearance of the earlier Handbook on Project Scheduling (cf. Węglarz 1999, Neumann and Zimmermann 1999), a large number of new results on resource–constrained project scheduling with schedule–dependent time windows have been published (see, e.g., Neumann et al 2000, Neumann and

Schwindt 2002, Neumann and Zimmermann 2002, Demeulemeester and Herroelen 2002, Neumann et al 2002b, 2003a,b, Schwindt and Trautmann 2003, Selle and Zimmermann 2003, Gentner et al 2004, Mellentien et al 2004, Neumann et al 2005, Schwindt 2005). These new results concern structural questions, a classification of time–based, financial and resource–based objective functions, different types of resources important in practice, efficient solution procedures, and new applications.

This chapter gives an overview of those new results on deterministic resource–constrained project scheduling with minimum and maximum time lags between activities. In Section 16.2, the basic project scheduling problem is formulated. Section 16.3 presents an order–based structural analysis of the feasible region of the basic project scheduling problem. Moreover, the classes of regular, convexifiable, locally regular, and locally concave objective functions are introduced and examples of such functions that are important to practice are given. Approaches to solving time–constrained project scheduling problems (i.e., without resource constraints) with the different types of objective functions introduced in Section 16.3 are discussed in Section 16.4. Section 16.5 deals with different types of resources important in practice and the resolution of so–called resource conflicts. In particular, renewable, cumulative (or storage), synchronizing, allocatable, and changeover resources are studied. In Section 15.6, we are concerned with new applications, such as factory pick–up of new cars and batch scheduling in process industries. Section 15.7 presents some conclusions and possible future research.

15.2 Basic project scheduling problem

We consider a project consisting of n *activities* $1, \dots, n$. Let $p_i \in \mathbb{N}$ be the *duration* or *processing time* of activity i, which is assumed to be carried out without interruption. In addition, we introduce the *fictitious activities* 0 and $n+1$ representing the beginning and completion, respectively, of the project, where $p_0 = p_{n+1} = 0$. Then $V = \{0, 1, \dots, n+1\}$ is the set of all activities.

Let $S_i \geq 0$ be the *start time* of activity $i \in V$, where $S_0 := 0$ (i.e., the project always begins at time zero). Then S_{n+1} represents the *project duration* or *makespan*. We assume that $S_{n+1} \leq \bar{d}$ where $\bar{d} \in \mathbb{N}$ is a prescribed maximum project duration or planning horizon. A sequence $S = (S_0, S_1, \dots, S_{n+1})$ with $S_i \geq 0$ $(i \in V)$ and $S_0 = 0$ is called a *schedule*.

A *minimum time lag* $d_{ij}^{min} \in \mathbb{Z}_{\geq 0}$ or *maximum time lag* $d_{ij}^{max} \in \mathbb{Z}_{\geq 0}$ can be prescribed between the start of two different activities i and j, that is, $S_j - S_i \geq d_{ij}^{min}$ or $S_j - S_i \leq d_{ij}^{max}$, respectively. If $d_{ij}^{min} = p_i$, $S_j - S_i \geq d_{ij}^{min}$ represents a *precedence constraint*. To ensure that the project is terminated by time $\bar{d}$, we introduce the maximum time lag $d_{0,n+1}^{max} := \bar{d}$.

It is well–known that an *activity–on–node network* N can be uniquely assigned to the project in question (cf. Neumann et al 2003a, Section 1.2). To do so we identify the activities $i \in V$ with the nodes of network N. If there is a minimum time lag d_{ij}^{min}, we introduce an arc $\langle i,j\rangle$ with weight $\delta_{ij} := d_{ij}^{min}$. If there is a maximum time lag d_{ij}^{max}, we introduce an arc $\langle j,i\rangle$ with weight $\delta_{ji} := -d_{ij}^{max}$. Due to maximum time lags, network N generally contains cycles. Let E be the set of arcs of network N. Then the above inequalities for the minimum and maximum time lags can be summarized as

$$S_j - S_i \geq \delta_{ij} \quad (\langle i,j\rangle \in E) \tag{15.1}$$

representing the so–called *temporal constraints.* A schedule S that satisfies the temporal constraints (15.1) is termed *time–feasible.* The set of time–feasible schedules is denoted by $\mathcal{S}_T$. It holds that $\mathcal{S}_T \neq \emptyset$ exactly if network N does not contain any cycle of positive length.

Assume that a set $\mathcal{R}^\rho$ of *renewable resources* (e.g., machines, manpower, or equipment) are required for carrying out the activities of the underlying project. Resource types different from renewable ones will be discussed in Section 16.5. Let $R_k \in \mathbb{N}$ be the capacity of renewable resource k available and $r_{ik} \in \{0, 1, \ldots, R_k\}$ be the amount of resource k used by activity i. Given a schedule $S = (S_i)_{i \in V}$,

$$\mathcal{A}(S,t) := \{i \in V \mid S_i \leq t < S_i + p_i\}$$

is the set of activities in progress, also called the *active set,* at time $t \in [0, \bar{d}]$.

$$r_k(S,t) := \sum_{i \in \mathcal{A}(S,t)} r_{ik}$$

is the amount of resource $k \in \mathcal{R}^\rho$ used at time $t \in [0, \bar{d}]$ given schedule S. Then the *(renewable–)resource constraints* are

$$r_k(S,t) \leq R_k \quad (k \in \mathcal{R}^\rho,\ 0 \leq t \leq \bar{d}) \tag{15.2}$$

A schedule S that satisfies (15.2) is called *(renewable–)resource–feasible.* The *resource profile* $r_k(S,\cdot)$ represents a step function on $[0, \bar{d}]$ continuous from the right. A schedule which is both resource– and time–feasible is termed *feasible.* The set of feasible schedules is denoted by $\mathcal{S}$.

The *basic project scheduling problem* consists of minimizing some objective function $f : \mathbb{R}_{\geq 0}^{n+2} \to \mathbb{R}$ on the set $\mathcal{S}$ of feasible schedules. In detail, this problem reads as follows:

$$\left.\begin{array}{lll} \text{Minimize} & f(S) & \\ \text{subject to} & S_j - S_i \geq \delta_{ij} & (\langle i,j\rangle \in E) \\ & S_i \geq 0 & (i \in V) \\ & S_0 = 0 & \\ & r_k(S,t) \leq R_k & (k \in \mathcal{R}^p,\ 0 \leq t \leq \bar{d}) \end{array}\right\} \quad \text{(P)}$$

In (P) we may delete the constraints $S_i \geq 0$ $(i \in V)$ because they are satisfied automatically. A feasible schedule S that minimizes function f on the feasible region $\mathcal{S}$ of (P) is called *optimal.* We assume that f is lower semicontinuous, i.e., $f(S) \leq \liminf_{S' \to S} f(S')$ for all schedules S. Since $\mathcal{S}$ is compact, the lower semicontinuity of f ensures that f attains its minimum on $\mathcal{S}$ provided that $\mathcal{S} \neq \emptyset$.

If we delete the resource constraints (15.2) from problem (P), i.e., we want to minimize function f on the set $\mathcal{S}_T$ of time–feasible schedules, we speak of the *resource relaxation* of (P) denoted by (RP). An optimal solution to problem (RP) is referred to as a *time–optimal schedule.*

15.3 Structural issues

In Section 16.3, we present an oder–based structural analysis of the feasible region $\mathcal{S}$ of problem (P) and a classification of objectives functions f of (P) important in practice that will be exploited for constructing efficient solution procedures for problems (RP) and (P) in Sections 16.4 and 16.5, respectively.

15.3.1 Properties of the feasible region $\mathcal{S}$

In this subsection, we often follow Neumann et al (2002b) closely. Whereas the set $\mathcal{S}_T$ of time–feasible schedules represents a (convex) polytope, the set $\mathcal{S}$ of feasible schedules is generally disconnected and represents the union of finitely many polytopes. The decision problem whether or not $\mathcal{S} \neq \emptyset$ is NP-complete (cf. Bartusch et al 1988).

Let $O \subset V \times V$ be a *strict order* (i.e., an asymmetric and transitive binary relation) in activity set V.

$$\mathcal{S}_T(O) := \{S \in \mathcal{S}_T \mid S_j \geq S_i + p_i \text{ for all } (i,j) \in O\}$$

is called the *order polytope* of O. Trivially, for the empty strict order $O = \emptyset$ we have $\mathcal{S}_T(\emptyset) = \mathcal{S}_T$. Strict order O is termed *time–feasible* if $\mathcal{S}_T(O) \neq \emptyset$ and *feasible* if $\emptyset \neq \mathcal{S}_T(O) \subseteq \mathcal{S}$. Order polytope $\mathcal{S}_T(O)$ is the set of all time–feasible schedules belonging to the *order network* $N(O)$. Network $N(O)$ results from the underlying project network N by adding arc $\langle i,j\rangle$ with weight p_i for each pair $(i,j) \in O$. If N already contains arc $\langle i,j \rangle$ with weight δ_{ij}, the latter weight is replaced by $\max(\delta_{ij}, p_i)$.

Recall that a feasible strict order O in V is termed *inclusion–minimal* if there is no feasible strict order O' in V with $O' \subset O$. The *basic structural theorem* first proved by Bartusch et al (1988) is then as follows:

Theorem 1. Let $\mathcal{O}$ be the (finite) set of all inclusion–minimal feasible strict orders in activity set V. Then $\mathcal{S} = \bigcup_{O \in \mathcal{O}} \mathcal{S}_T(O)$.

For a schedule $S \in \mathcal{S}_T$,

$$O(S) := \{(i,j) \in V \times V \mid i \neq j, S_j \geq S_i + p_i\}$$

is the strict order induced by S. The corresponding order polytope $\mathcal{S}_T(O(S))$ is called the *schedule polytope* of S and represents the set of all time–feasible schedules belonging to *schedule network* $N(O(S))$.

Theorem 1 gives a representation of feasible region $\mathcal{S}$ as union of finitely many order polytopes which generally overlap. Next, we consider a partition of $\mathcal{S}$ (into disjoint polytope–like sets). Given schedule $S \in \mathcal{S}_T$,

$$\mathcal{S}_T^{=}(O(S)) := \{S' \in \mathcal{S}_T(O(S)) \mid O(S') = O(S)\}$$

is the set of all schedules inducing the same strict order as S and is termed the *equal–order set* for S. $\mathcal{S}_T^{=}(O(S))$ represents a polytope generally without a part of its boundary. Since $S \in \mathcal{S}_T^{=}(O(S))$ for each $S \in \mathcal{S}$ and there are only finitely many distinct strict orders $O(S)$, we have $\mathcal{S} = \bigcup_{S \in \mathcal{S}} \mathcal{S}_T^{=}(O(S))$ and thus a finite partition of $\mathcal{S}$.

As we will see in Subsection 15.3.2, special points of $\mathcal{S}$ represent possible optimal schedules for problem (P). Let $\mathcal{M}$ be a compact subset of $\mathbb{R}^{n+2}$. Recall that $S \in \mathcal{M}$ is a *minimal point* (or *maximal point*) of $\mathcal{M}$ precisely if there is no $S' \in \mathcal{M}$, $S' \neq S$, with $S' \leq S$ (or $S' \geq S$, respectively) where $\leq$ is meant componentwise. $S \in \mathcal{M}$ is an *extreme point* of $\mathcal{M}$ exactly if S does not lie on a line segment that joins two other points of $\mathcal{M}$. $S \in \mathcal{M}$ is a *local minimizer* of objective function f on $\mathcal{M}$ exactly if for some $\varepsilon > 0$, S minimizes f on $\mathcal{M} \cap B_\varepsilon(S)$ with $B_\varepsilon(S) := \{S' \in \mathbb{R}^{n+2}_{\geq 0} \mid \|S' - S\| < \varepsilon\}$.

15.3.2 Different types of objective functions

In this subsection, we present a classification of lower semicontinuous objective functions f important in practice, give examples of the different function types, and state which special points of the feasible region $\mathcal{S}$ are candidates for optimal schedules (cf. Neumann et al 2003a, Section 3.3, and Schwindt 2005, Section 2.3).

Objective function f is called *regular* if f is nondecreasing, i.e., $S \leq S'$ implies $f(S) \leq f(S')$. Obviously, there is a minimal point of $\mathcal{S} \neq \emptyset$ which represents an optimal schedule for problem (P) with regular f. Moreover, the unique minimal point of any order polytope $\mathcal{S}_T(O)$ is the minimizer of f on $\mathcal{S}_T(O)$. This minimal point coincides with the *earliest schedule* $ES = (ES_i)_{i\in V}$, where the earliest start time ES_i of activity i equals the longest path length from node 0 to node i in order network $N(O)$, which can be found in polynomial time.

Examples of regular objective functions are (i) the *project duration* or *makespan* S_{n+1}, (ii) the *maximum lateness* $\max_{i\in V}(S_i + p_i - d_i)$, where $d_i \in \mathbb{N}$ is a prescribed due date for activity i, (iii) the *weighted flow time* $\sum_{i\in V} w_i^F(S_i+p_i-r_i)$, where $r_i \in \mathbb{Z}_{\geq 0}$ is a given ready time of activity i and $w_i^F \geq 0$ is a weighting factor describing the importance or urgency of activity i, and (iv) the *weighted tardiness* $\sum_{i\in V} w_i^T(S_i + p_i - d_i)^+$, where $z^+ = \max(z, 0)$ and $w_i^T \geq 0$ is the tardiness cost of activity i per unit time.

The *weighted earliness* $\sum_{i\in V} w_i^E(d_i - S_i - p_i)^+$ with $w_i^E \geq 0$ represents a so–called *antiregular* or nonincreasing objective function f, i.e., $S \leq S'$ implies $f(S) \geq f(S')$. There is always a maximal point of $\mathcal{S} \neq \emptyset$ which is an optimal schedule for (P) with antiregular f.

As mentioned above, the minimizer of a regular function f (or an antiregular function) on order polytope $\mathcal{S}_T(O)$ can be computed in polynomial time. If function f is convex and (due to the lower semicontinuity) continuous, a minimizer of f on $\mathcal{S}_T(O)$ can be found in polynomial time as well, e.g., by the ellipsoid method (cf. Grötschel et al 1995) or, generally more efficiently, by interior–point methods provided that some mild technical assumptions are satisfied.

The more general convexifiable functions admit a smooth coordinate tranformation such that the resulting resource relaxation (RP) represents a convex optimization problem with linear constraints. Objective function $f : \mathcal{S}_T \rightarrow \mathbb{R}$ is called *convexifiable* if there exists a bijection $\varphi : \mathcal{S}_T \rightarrow X$, where φ and φ^{-1} are continuously differentiable and X is some Euclidean space, such that $f\circ\varphi^{-1}$ is a convex function and the images $\varphi(\mathcal{S}_T(O)) := \{\varphi(S) \mid S \in \mathcal{S}_T(O)\}$ of all order polytopes by φ are convex sets. If $f\circ\varphi^{-1}$ is linear, f is termed a *linearizable* function. Note that all sets $\varphi(\mathcal{S}_T(O))$ are compact and set $X = \varphi(\mathcal{S}_T)$ is convex. For a convexifiable function f, there is always a local minimizer of f on some order polytope $\mathcal{S}_T(O)$ with $O \in \mathcal{O}$ that minimizes f on $\mathcal{S} \neq \emptyset$ (cf. Schwindt 2005, Subsection 2.3.1). If f is linearizable, there is always an extreme point (or vertex) of $\mathcal{S}_T(O)$ that minimizes f on $\mathcal{S}_T(O)$.

Obviously, each linear function is linearizable and each convex function is convexifiable. Minimizing a *linear function* $\sum_{i\in V} v_i(S_i + p_i)$ with $v_i \in \mathbb{R}$, where we may omit the additive constant $\sum_{i\in V} v_i p_i$, says that for $v_i > 0$ (or $v_i < 0$), activity i should be completed as early (or as late, respectively) as possible.

The *weighted earliness–tardiness* $\sum_{i\in V}[w_i^E(d_i - S_i - p_i)^+ + w_i^T(S_i + p_i - d_i)^+]$ is convex with respect to S and is important to just–in–time production, where we strive for avoiding poorly timing of the beginning of activities. Also, the weighted earliness–tardiness is used for *rescheduling* in the following sense. In practice, input data such as the amount of resources available and ready or processing times of activities are often subject to change. For example, machines may break down or workers are late or absent. A consequence of such disturbances is that a previously feasible *baseline schedule* often becomes infeasible. We then wish to find a new feasible schedule as close as possible to the old one because larger deviations generally cause difficulties in practice. This can be done by solving a project scheduling problem whose objective function is the weighted earliness–tardiness, where the due dates d_i equal the completion times of the activities i in the previous baseline schedule.

The *negative net present value* of a project $-\sum_{i\in V} c_i^F \beta^{S_i+p_i}$ represents a linearizable objective function (cf. Schwindt 2005, Subsection 2.3.1). Here $c_i^F \in \mathbb{R}$ is the cash flow associated with activity i that may be positive (i.e., a payment received) or negative (i.e., a disbursement incurred) and is supposed to occur at its completion time $S_i + p_i$, and β with $0 < \beta \leq 1$ is the discount rate.

The following two types of objective functions are important to resource–based objectives. A (lower semicontinuous) objective function f is called *locally regular* (or *locally concave*) if f is regular (or concave, respectively) on all equal–order sets $\mathcal{S}_T^=(O(S)), S \in \mathcal{S}$. For a locally regular (or locally concave) function f, there is always a minimal point (or extreme point, respectively) S of some schedule polytope $\mathcal{S}_T(O(S))$ which is a minimizer of f on $\mathcal{S} \neq \emptyset$ (cf. Neumann et al 2003a, Subsections 3.3.7 and 3.3.8).

The *total procurement cost* $\sum_{k\in\mathcal{R}^\rho} c_k \max_{0\leq t\leq \overline{d}} r_k(S,t)$, where $c_k \geq 0$ is the procurement cost per unit of resource k, represents a locally regular function, which is lower semicontinuous but not necessarily continuous. The project scheduling problem (P) or (RP) with the total procurement cost as objective functions is also referred to as the *resource investment problem*.

The *total squared utilization cost* $\sum_{k\in\mathcal{R}^\rho} c_k \int_0^{\overline{d}} r_k^2(S,t)dt$, where $c_k \geq 0$ is the cost incurred per unit of resource k and per unit time, is a locally concave function. Another locally concave function is the *total overload cost* $\sum_{k\in\mathcal{R}^\rho} c_k \int_0^{\overline{d}} [r_k(S,t) - Y_k]^+ dt$ where $Y_k \in \mathbb{N}$ is a given supply or a threshold for the resource utilization of resource k. If no threshold is given, Y_k can be replaced by the (rounded) average resource utilization $\lceil \sum_{i\in V} r_{ik}p_i/\overline{d} \rceil$. The latter two objective functions are important to *resource levelling*, where we want to utilize the resources evenly over time. Additional objective functions related to resource levelling, which are locally concave, can be found in Neumann et al (2003a, Section 3.1) and Demeulemeester and Herroelen (2002, Subsection 3.1.3).

In practice, resources required for carrying out project activities are often rented instead of purchased (e.g., expensive machinery), which incurs fixed and variable costs. For each unit of resource $k \in \mathcal{R}^\rho$ rented, there are a *fixed renting cost* $c_k^f \geq 0$ arising when bringing the unit into service and a *variable renting cost* $c_k^v \geq 0$ referring to one unit of resource k and one unit of time for which the resource unit is rented.

Given schedule S, let $\varphi_k(S,t)$ (or φ_{kt}, for short) be the amount of resource k rented at time $t \in [0, \overline{d}]$. Each resource is brought into service only finitely many times within planning period $[0, \overline{d}]$. Thus, we can restrict ourselves to right continuous step functions $\varphi_k(S, \cdot)$ with finitely many jump discontinuities. Given $\varphi_k(S, \cdot)$, $c_k^v \int_0^{\overline{d}} \varphi_k(S,t)dt$ is the *total variable renting cost* for resource k. Let J_k be the set of jump discontinuities of $\varphi_k(S, \cdot)$ on $[0, \overline{d}]$ (including point in time 0 if $\varphi_k(S, 0) > 0$) and let $\Delta^+\varphi_{kt} \geq 0$ be the increase in the amount of resource k rented at time $t \in J_k$. Then $c_k^f \sum_{t \in J_k} \Delta^+ \varphi_{kt}$ is the *total fixed renting cost* for resource k. *Renting policy* $\varphi(S, \cdot) = (\varphi_k(S, \cdot))_{k \in \mathcal{R}^\rho}$ is called *feasible* (with respect to schedule S) if

$$\varphi_k(S,t) \geq r_k(S,t) \text{ for all } k \in \mathcal{R}^\rho \text{ and } t \in [0, \overline{d}]$$

and is termed *optimal* if it is feasible and minimizes the corresponding *total renting cost* $\sum_{k \in \mathcal{R}^\rho} [c_k^v \int_0^{\overline{d}} \varphi_k(S,t)dt + c_k^f \sum_{t \in J_k} \Delta^+ \varphi_{kt}]$. The objective function f of the *resource renting problem* represents the total renting cost belonging to an optimal renting policy and is given by

$$f(S) := \sum_{k \in \mathcal{R}^\rho} \min_{\varphi_k(S,\cdot) \geq r_k(S,\cdot)} \left[c_k^v \int_0^{\overline{d}} \varphi_k(S,t)dt + c_k^f \sum_{t \in J_k} \Delta^+ \varphi_{kt} \right] \qquad (15.3)$$

It can be shown that function f is lower semicontinuous (cf. Neumann et al 2003a, Subsection 3.3.8) and locally concave (cf. Nübel 2001).

For regular (and antiregular) as well as convexifiable objective functions f, the resource relaxation (RP) can be solved in an efficient way. The same is true for the computation of a minimizer of f on an order polytope $\mathcal{S}_T(O)$ for any strict order O in activity set V. Thus, it is expedient to use the following *relaxation–based approach* to solving problem (P) with regular, antiregular, or convexifiable f. As will be discussed in Section 16.5 in more detail, resource conflicts (i.e., violations of the resource constraints (15.2)) can be resolved by introducing precedence constraints. The solving of problem (P) can then be replaced by solving a sequence of problems of type (RP) with $\mathcal{S}_T(O)$ for some strict order O in V instead of $\mathcal{S}_T$.

For locally regular and locally concave objective functions f, even the resource relaxation (RP) is generally NP–hard. Hence, the relaxation–based approach is not recommended. Instead, a so–called *tree–based approach* is proposed. As mentioned above, minimal or extreme points (i.e., vertices) of schedule polytopes $\mathcal{S}_T(O(S)), S \in \mathcal{S}$, represent candidates for optimal schedules for locally regular or locally concave f. A vertex of polytope $\mathcal{S}_T(O(S))$ corresponds to a spanning tree of schedule network $N(O(S))$. The $n+1$ arcs of such a spanning tree T, say arcs $\langle i,j\rangle \in E^T$ with weights δ_{ij}^T, correspond to $n+1$ linearly independent binding temporal constraints $S_j - S_i = \delta_{ij}^T$ ($\langle i,j\rangle \in E^T$), which together with $S_0 = 0$ have a unique solution, namely the vertex in question. Therefore, to construct a spanning tree of $N(O(S))$, we consecutively fix start times of activities such that, step by step, temporal constraints $S_j - S_i \geq \delta_{ij}^T$ become binding.

15.4 Time–constrained project scheduling

In this section, we discuss approaches to solving time–constrained project scheduling problems with different types of objective functions. In particular, we consider regular, convexifiable, locally regular, and locally concave objective functions. The problem under consideration is

$$\left.\begin{array}{ll}\text{Minimize} & f(S)\\ \text{subject to} & S \in \mathcal{S}_T\end{array}\right\} \quad \text{(RP)}$$

and represents the resource relaxation of our basic project scheduling problem (P) briefly discussed in Sections 16.2 and 16.3.

15.4.1 Regular and convexifiable objective functions

For problem (RP) with regular objective function, an optimal solution is always given by the *earliest schedule ES* (cf. Subsection 15.3.2). Schedule ES can be found by a so–called label–correcting algorithm with time complexity $O(|V||E|)$, cf. Ahuja et al (1993, Section 5.4). Convexifiable objective functions like linear, weighted earliness–tardiness, or negative net present value functions are considered e.g. in Mellentien et al (2004), Neumann et al (2003a, Subsections 3.5.2 and 3.5.3), and Schwindt (2005, Section 3.2). In what follows, we show that the dual of the linear problem represents a minimum cost flow problem and thus can be solved very efficiently (cf. Russell, 1970). Furthermore, we present a steepest descent algorithm for the time–constrained earliness–tardiness problem, which can also be used to solve the time–constrained net present value problem.

15.4.1.1 Linear objective function. Given a real–valued weight v_i for each activity $i \in V$, the linear problem with objective function $\sum_{i\in V} v_i S_i$ can be written as

$$\left.\begin{array}{ll} \text{Maximize} & \sum_{i\in V} w_i S_i \\ \text{subject to} & S_i - S_j \leq -\delta_{ij} \quad (\langle i,j\rangle \in E) \\ & S_0 = 0 \end{array}\right\} \quad \text{(WSTP)}$$

where $w_i := -v_i$ $(i \in V)$. The dual problem of (WSTP) is

$$\left.\begin{array}{lll} \text{Minimize} & \sum\limits_{\langle i,j\rangle\in E} -\delta_{ij} x_{ij} & \\ \text{subject to} & \sum\limits_{\langle h,i\rangle\in E} x_{hi} - \sum\limits_{\langle i,j\rangle\in E} x_{ij} = w_i & (i \in V) \\ & x_{ij} \geq 0 & (\langle i,j\rangle \in E) \end{array}\right\} \quad \text{(DWSTP)}$$

Note that equation $S_0 = 0$ gives rise to an unrestricted real–valued variable in the flow balance constraint for activity 0. Since the corresponding coefficient in the objective function is equal to zero, we may omit the variable and put $w_0 := -\sum_{i\in V\setminus\{0\}} w_i$, which means that $\sum_{i\in V} w_i = 0$. Problem (DWSTP) then represents a *minimum–cost flow problem* in project network N with unit costs $-\delta_{ij}$ and infinite arc capacities as well as supplies $-w_i$ at nodes $i \in V$. Such a network flow problem can be solved quite efficiently by some polynomial–time network flow algorithm, see e.g., Ahuja et al (1993, Chapter 10) and Goldberg (1997). Based on a minimum–cost flow x, a time–optimal schedule S can readily be constructed by exploiting the complementary slackness conditions, from which it follows that $S_j - S_i = \delta_{ij}$ for all $\langle i,j\rangle \in E$ with $x_{ij} > 0$.

15.4.1.2 Weighted earliness–tardiness. Exact solution procedures for the time–constrained earliness–tardiness problem, i.e.

$$\begin{array}{ll} \text{Minimize} & f(S) = \sum\limits_{i\in V}[w_i^E(d_i - S_i - p_i)^+ + w_i^T(S_i + p_i - d_i)^+] \\ \text{subject to} & S \in \mathcal{S}_T \end{array}$$

have been devised by Schwindt (2000) and for the case of precedence constraints by Vanhoucke et al (2001). We briefly sketch Schwindt's steepest descent approach, as presented in Neumann et al (2003a, Subsection 3.5.3). The algorithm seems to be the most efficient solution method known thus far.

The objective function f of the earliness–tardiness problem is continuous, convex, but not differentiable at points S with $S_i = d_i - p_i$ for some $i \in V$. The left–hand partial derivative of function f with respect to S_i at schedule S equals $-w_i^E$ if $S_i + p_i \leq d_i$ and equals w_i^T, otherwise. Analogously, the right–hand partial derivative equals w_i^T, if $S_i + p_i \geq d_i$, and $-w_i^E$, otherwise.

For the directional derivative of f at S in direction z, we obtain $\partial f/\partial z(S) = \sum_{i\in V} \partial f_i/\partial z(S)$ with

$$\frac{\partial f_i}{\partial z}(S) := \begin{cases} -w_i^E z_i, & \text{if either } S_i + p_i < d_i \text{ or both} \\ & S_i + p_i = d_i \text{ and } z_i \leq 0 \\ w_i^T z_i, & \text{otherwise} \end{cases}$$

Given schedule $S \in \mathcal{S}_T$, let $E^T \subseteq E$ be a set of corresponding linearly independent binding temporal constraints, which can be represented by a spanning forest T of project network N with arc set E^T (cf. Subsection 15.3.2). Then a *steepest descent direction* z at schedule S can be found by solving the following *steepest descent direction problem*

$$\left.\begin{array}{lll} \text{Minimize} & g(z) := \partial f/\partial z(S) & \\ \text{subject to} & z_j - z_i \geq 0 & (\langle i,j\rangle \in E^T) \\ & z_0 = 0 & \\ & -1 \leq z_i \leq 1 & (i \in V) \end{array}\right\} \quad \text{(SDD)}$$

If $z = 0$ is an optimal solution to (SDD), S represents a local (and global) minimizer of f. Since g is piecewise linear and the coefficient matrix of (SDD) is totally unimodular, restrictions $-1 \leq z_i \leq 1$ can be replaced by $z_i \in \{-1, 0, 1\}$. Moreover, starting with the earliest schedule ES, the sequence of schedules generated by the proposed steepest descent algorithm is componentwise nondecreasing, i.e., we can focus on steepest descent directions $z \in \{0,1\}^{n+2}$, which can be determined as follows. Let g_i be the right–hand partial derivative of f with respect to S_i at S. If there is a source i of spanning forest T with $g_i \geq 0$, then there exists an optimal solution z to (SDD) with $z_i = 0$, and node i and all incident arcs $\langle i,j\rangle$ are eliminated from T. If there is a sink j of spanning forest T with $g_j < 0$, then $z_j = 1$ holds for any optimal solution to (SDD), and node j and all incident arcs $\langle i,j\rangle$ are eliminated from T. Otherwise, T contains a source i with at most one successor j (and $g_i < 0$) or a sink j with exactly one predecessor i (and $g_j \geq 0$). In both cases, activity i is shifted exactly if activity j is shifted, i.e., $z_i = z_j$. Thus, nodes (activities) i and j can be merged into an aggregate activity with partial derivative $g_i + g_j$. We perform those steps until all nodes aside from 0 have been deleted or have been merged with node 0, where from constraint $z_0 = 0$ it follows that $z_i = 0$ for all nodes i merged with node 0.

Given steepest descent direction $z \geq 0$, *stepsize* σ can be determined as follows. We stop moving from S in direction z if some activity i is completed at its due date d_i (i.e., $S_i + p_i = d_i$) or if some temporal constraint becomes binding. Thus, we have

$$\sigma := \min\Big[\min_{i\in V: S_i+p_i<d_i}(d_i - S_i - p_i), \min_{\langle i,j\rangle\in E: z_i>z_j}(S_j - S_i - \delta_{ij})\Big]$$

If for the resulting schedule $S' = S + \sigma z$ it holds that $z = 0$ is an optimal solution to the corresponding problem (SDD), then S' is an optimal schedule. Otherwise, we again perform a descending step in direction z' from point S'.

Note that each aggregate node j deleted from forest T when determining steepest descent direction z represents a weak component of forest T. Delaying any of those components j with $z_j = 1$ uniformly (until one of the corresponding activities meets its due date or a temporal constraint becomes binding), results in a decrease in the objective function value. Thus, we can improve the efficiency of the steepest descent procedure by performing a sequence of descent steps, where in each iteration we delay all (remaining) components j with $z_j = 1$ until one of those weak components is "connected" with node 0 by a new binding temporal constraint or an activity of some weak component is completed at its due date. The corresponding weak component is then omitted in the subsequent descend steps.

15.4.1.3 Net present value. The time–constrained net present value problem with objective function

$$f(S) = -\sum_{i \in V} c_i^F \beta^{S_i + p_i}$$

has been widely discussed in literature, cf. e.g. Herroelen et al (1997) and Neumann et al (2003a, Subsection 3.5.2). As has been stated in Subsection 15.3.2, f is linearizable and consequently there is an extreme point of $\mathcal{S}_T$ that minimizes f on $\mathcal{S}_T$. Moreover, f is binary–monotone (i.e. monotone in each binary direction $z \in \{-1, 0, 1\}^{n+2}$) and differentiable where the directional derivative of f at S in direction z is given by

$$\frac{\partial f}{\partial z}(S) = \sum_{i \in V} - \ln \beta \, c_i^f \beta^{S_i + p_i} z_i$$

As has been shown by Schwindt and Zimmermann (2001), the steepest descent approach proposed for the earliness–tardiness problem is also very efficient for the time–constrained net present value problem. Since $\partial f / \partial z(S)$ is a linear function in z, the corresponding problem (SDD) represents a linear program. Starting with the earliest schedule ES, there is again a steepest descent direction $z \in \{0, 1\}^{n+2}$, which can be determined as described above, where g_i is initialized with $- \ln \beta \, c_i^f \beta^{S_i + p_i}$.

Given a steepest descent direction $z \in \{0, 1\}^{n+2}$ at schedule $S \in \mathcal{S}_T$, binary–monotone function f is nonincreasing on the half–line emanating from S in direction z. Therefore,

$$\sigma := \min_{\langle i,j \rangle \in E : z_i > z_j} (S_j - S_i - \delta_{ij})$$

is the largest stepsize such that $S+\sigma z$ is time–feasible. This means that we delay all activities i with $z_i = 1$ until a new temporal constraint becomes binding. Thus, performing a sequence of descent steps until all weak components shifted are "connected" with node 0 again provides an extreme point of $\mathcal{S}_T$.

15.4.2 Locally regular functions

For locally regular objective functions f, Neumann and Zimmermann (1999, 2000) and Neumann et al (2000, 2003a, Section 3.6) have proposed a so–called tree–based approach. As mentioned in Subsection 15.3.2, minimal points of schedule polytopes represent candidates for optimal schedules for locally regular objective functions. A minimal point S of schedule polytope $\mathcal{S}_T(O(S))$ can be represented by a *spanning outtree* T of schedule network $N(O(S))$ with root 0, arc set E^T, and arc weights δ_{ij}^T, where S is the unique solution to the system of $n+2$ linear equations $S_0 = 0$, $S_j - S_i = \delta_{ij}^T$ $(\langle i,j\rangle \in E^T)$.

To enumerate the minimal points of all schedule polytopes $\mathcal{S}_T(O(S))$, we enumerate the corresponding spanning outtrees by consecutively fixing start times of activities. More precisely, let set Ω contain the pair (C, S^C) for each partial schedule (subtree) $S^C := (S_i)_{i\in C}$ already constructed, where we start with $C = \{0\}$ and $S_0 = 0$. In each iteration, we remove a pair (C, S^C) from Ω. If $C = V$, we have found a minimal point of some schedule polytope. Otherwise, we extend the partial schedule S^C as follows. For each $j^* \in V \setminus C$, we determine the set $\mathcal{D}_{j^*}$ of tentative start times $t \in [ES_{j^*}, LS_{j^*}]$ for which there is an activity $i \in C$ such that

(i) $t = S_i + \delta_{ij^*}$, i.e., temporal constraint $S_{j^*} - S_i \geq \delta_{ij^*}$ is binding or

(ii) $t = S_i + p_i$, i.e., precedence constraint $S_{j^*} \geq S_i + p_i$ is binding.

For each $t \in \mathcal{D}_{j^*}$, we then add the corresponding extended partial schedule $S^{C'}$ with $C' = C \cup \{j\}$ and $S_j = t$ to Ω. Next, we take a new pair (C, S^C) from Ω and proceed in the same way until all partial schedules from Ω have been investigated. The described tree–based enumeration scheme does not avoid that one and the same outtree (minimal point) is generated more then once. Conditions for pairs (i, j^*) that avoid the latter drawback can be found in Nübel (1999).

Tree–based branch–and–bound algorithms for the resource investment problem (cf. Subsection 15.3.2) can be found in Nübel (1999) and Zimmermann (2001, Section 5.3).

15.4.3 Locally concave functions

The tree–based enumeration scheme sketched in Subsection 15.4.2 can be generalized to locally concave objective functions f. As mentioned in Subsection 15.3.2, extreme points of schedule polytopes represent candidates for optimal schedules. Such an extreme point can be represented by a *spanning tree* T of schedule network $N(O(S))$. To enumerate all spanning trees and not only all spanning outtrees, we have to take into consideration the following four cases when determining sets $\mathcal{D}_{j^*}$:

(i) $t = S_i + \delta_{ij^*}$, i.e., temporal constraint $S_{j^*} - S_i \geq \delta_{ij^*}$ is binding or

(ii) $t = S_i - \delta_{j^*i}$, i.e., temporal constraint $S_i - S_{j^*} \geq \delta_{j^*i}$ is binding or

(iii) $t = S_i + p_i$, i.e., precedence constraint $S_{j^*} - S_i \geq p_i$ is binding or

(iv) $t = S_i - p_{j^*}$, i.e., precedence constraint $S_i - S_{j^*} \geq p_{j^*}$ is binding.

Branch–and–bound procedures exploiting the tree–based enumeration scheme for resource levelling problems with objective functions total squared utilization cost or total overload cost can be found in Zimmermann (2001, Section 5.3) and Neumann et al (2003, Subsection 3.6.2). For the locally concave objective function

$$f(S) = \sum_{k \in \mathcal{R}^\rho} \min_{\varphi_k(S,\cdot) \geq r_k(S,\cdot)} \left[c_k^v \int_0^{\overline{d}} \varphi_k(S,t)\, dt + c_k^f \sum_{t \in J_k} \Delta^+ \varphi_{kt} \right]$$

of the *resource renting problem*, the proposed tree–based approach can be used within the framework of a branch–and–bound procedure, cf. Nübel (2001) and Neumann et al (2003a, Subsection 3.6.2). In this case, an optimal renting policy and the corresponding total renting cost must be computed for each partial schedule constructed by the tree–based enumeration scheme.

Recall that $\varphi_k(S,t)$ is the amount of resource k rented at time t given schedule S and $\Delta^+\varphi_{kt}$ denotes the increase in the amount of resource k rented at jump point $t \in J_k$ (cf. Subsection 15.3.2). Moreover, let $\Delta^-\varphi_{kt} \geq 0$ be the decrease in the amount of resource k rented at time $t \in J_k$. An *optimal renting policy* $\varphi^*(S,\cdot) = (\varphi_k^*(S,\cdot))_{k \in \mathcal{R}^\rho}$ for given schedule S can be found as follows. If $c_k^v = 0$ for some $k \in \mathcal{R}^\rho$, it is optimal to rent the maximum amount of resource k required at the beginning of the project and to release this amount at the project completion. This means that $\varphi_k^*(S,t) = \max_{\tau \in [0,\overline{d}]} r_k(S,\tau)$ for all $t \in [0,\overline{d}]$. For $c_k^f = 0$, it is optimal to rent the required resource amount for each activity $i \in V$ at its start time and to release it at its completion time, i.e., $\varphi_k^*(S,t) = r_k(S,t)$ for all $t \in [0,\overline{d}]$.

Now let $k \in \mathcal{R}^\rho$ be a resource with $c_k^v > 0$ and $c_k^f > 0$. In this case, $\varphi_k(S,\cdot) = r_k(S,\cdot)$ generally does not represent an optimal renting policy because it may be advantageous to rent resource units at points in time where they are not required for schedule S. Moreover, it cannot be optimal to increase or decrease the amount of resource k rented at points in time at which no activity is started or completed, respectively. Thus, it holds that

$$\Delta^-\varphi_{kt}^* = 0 \text{ for all } t \notin CT, \quad \Delta^+\varphi_{kt}^* = 0 \text{ for all } t \notin ST$$

where ST and CT are the sets of all start times and completion times, respectively, of activities $i \in V$. Hence, the step functions $\varphi_k^*(S, \cdot)$ are well–defined by specifying $\varphi_k^*(S, t)$ for all $k \in \mathcal{R}^\rho$ and all $t \in DT := ST \cup CT$.

For computing an optimal renting policy $\varphi_k^*(S, \cdot)$ for resource k, we start with the renting policy given by $\varphi_k(S, t) := r_k(S, t)$ for all $t \in DT$ and iterate the decision times $t \in DT$ where new resource units are rented (i.e., where $\Delta^+ \varphi_{kt} > 0$) in increasing order. In each iteration, we check whether the total renting cost for resource k can be decreased by extending the renting of $\min(\Delta^- \varphi_{kt'}, \Delta^+ \varphi_{kt})$ units of k that had been released on an earlier decision time t', where decision times $t' < t$ are examined in decreasing order. Each time we have identified such a cost saving, we update renting policy $\varphi_k(S, t)$ accordingly.

15.5 Resource–constrained project scheduling

When coping with real–life projects, various kinds of resource constraints may occur. In this section, we are concerned with algorithms for scheduling projects with different types of scarce resources. In Subsection 15.5.1 we review relaxation–based and tree–based algorithms for the basic project scheduling problem (P) with renewable–resource constraints using the methods for time–constrained project scheduling discussed in Section 16.4. In Subsections 15.5.2 to 15.5.4, we show how to adapt the relaxation–based approach to problems involving allocatable, synchronizing, changeover, and cumulative resources. Those new types of resources are needed when dealing with new applications discussed in Section 15.6.

15.5.1 Renewable resources

15.5.1.1 Regular and convexifiable objective functions. We consider a time–optimal schedule S that results from applying the minimum–cost flow or steepest descent methods to the resource relaxation (RP) of basic project scheduling problem (P) with regular or convexifiable objective function f. For such a schedule S, there generally exist points in time t where the amount of some resource $k \in \mathcal{R}^\rho$ used at time t exceeds the resource capacity R_k. If such a *resource conflict* occurs, not all activities from active set $\mathcal{A}(S, t)$ can be processed at the same time. A set F of activities that cannot be executed simultaneously because

$$\sum_{i \in F} r_{ik} > R_k \text{ for some } k \in \mathcal{R}^\rho$$

is called a *forbidden set*. It has been shown by Bartusch et al (1988) that a schedule S is resource–feasible precisely if for any inclusion–minimal forbidden set F there exist two activities i, j such that activity j is started at the earliest once activity i has been completed, i.e., $S_j \geq S_i + p_i$ (we then say that forbidden set F has been broken up). This means that in case of a resource conflict at time t we have to partition forbidden active set $\mathcal{A}(S,t)$ into two sets A and B such that B contains an activity j from each inclusion–minimal forbidden set $F \subseteq \mathcal{A}(S,t)$. If we then introduce the precedence constraints

$$S_j - S_i \geq p_i \quad (j \in B) \tag{15.4}$$

between some activity $i \in A$ and all activities $j \in B$, we break up all inclusion–minimal forbidden sets $F \subseteq \mathcal{A}(S,t)$ containing activity $i \in A$. Since this property holds for any partition $\{A, B\}$ of $\mathcal{A}(S,t)$, without loss of generality we may restrict ourselves to inclusion–minimal sets B, which are referred to as *minimal delaying alternatives* in literature. An efficient recursive procedure for computing all minimal delaying alternatives for a given forbidden set F can be found in Neumann et al (2003a, Subsection 2.5.1). We note that set A is a non–forbidden set and thus all activities $i \in A$ may be processed jointly.

The relaxation–based approach to the scheduling of projects with renewable resources and convexifiable objective function f is now as follows. We start by computing a time–optimal schedule S for the resource relaxation (RP). If S is resource–feasible, we have found an optimal schedule. Otherwise, we determine some activity start time t where schedule S causes a resource conflict. We then compute a minimal delaying alternative B for forbidden active set $\mathcal{A}(S,t)$ and refine relaxation (RP) by the corresponding precedence constraints (15.4) for some $i \in \mathcal{A}(S,t) \setminus B$. Subsequently, we re–perform the time–constrained project scheduling, which either shows the refined relaxation to be unsolvable because we have generated a cycle of positive length in the augmented project network $N = N(O)$ with $O = \{(i,j) \mid j \in B\}$, or which yields a new schedule S. We re–iterate these steps until we have reached a deadlock (i.e., N contains a cycle of positive length) or a feasible schedule S has been found. This schedule-generation scheme is expanded to the enumeration scheme of a branch–and–bound algorithm if in each iteration, we branch over all pairs (i, B) for which B is a minimal delaying alternative for set $\mathcal{A}(S,t)$ and i is some activity from set $A = \mathcal{A}(S,t) \setminus B$.

The earliest branch–and–bound algorithm based on this enumeration scheme was proposed by Icmeli and Erengüç (1996) for the resource–constrained net present value problem with precedence constraints. Later on, De Reyck and Herroelen (1998), Schwindt (2000), and Neumann and Zimmermann (2002) have devised branch–and–bound methods for the resource–constrained makespan, earliness–tardiness, and net present value problems with general temporal constraints. In the algorithms by Schwindt and by Neumann and Zimmermann, the re–optimization of schedule S after having added new precedence constraints of type (15.4) is performed by using a dual flattest ascent method.

For solving problem (P) with regular objective function, Franck et al (2001) have proposed to break up forbidden active sets $\mathcal{A}(S,t)$ by adding *disjunctive precedence constraints*

$$S_j - \min_{i\in A}(S_i + p_i) \geq 0 \quad (j \in B) \tag{15.5}$$

instead of the ordinary precedence constraints (15.4). Conditions (15.5), which represent the disjunction of precedence constraints (15.4) over different choices of activity i, say that no activity j from minimal delaying alternative B can be started before the first activity i from set A has been completed. Accordingly, in an iteration of the enumeration scheme, only one child node has to be generated per minimal delaying alternative B. Solving the refined resource relaxation with disjunctive precedence constraints and regular objective function can be done in pseudopolynomial time, which is linear in the maximum project duration $\overline{d}$.

Based on the enumeration scheme with disjunctive precedence constraints, Franck et al (2001) have developed three different types of *truncated branch–and–bound algorithms* for problem (P) with the makespan objective function. Those algorithms restrict the search for an optimal schedule to a part of the enumeration tree. The so–called *performance–guarantee algorithm* with maximum relative error $\varepsilon > 0$ of the objective function value only considers enumeration nodes for further branching whose lower bounds are more than $\varepsilon \cdot 100\,\%$ below the current upper bound. In the *filtered beam search method*, the number of child nodes added to the enumeration tree is limited to a given maximum number. Those child nodes are selected in two steps. At first, applying a simple filter criterion establishes a pre–selection of nodes, among which a more elaborate beam criterion identifies the most promising nodes to branch from. The *decomposition method* exploits the property that there exists a feasible schedule precisely if there is a feasible subschedule for each strong component of the project network. The method consists of two phases during which the strong components are scheduled separately and a feasible schedule for the entire project is determined based on the subschedules obtained for the strong components.

15.5.1.2 Locally regular and locally concave objective functions.

The tree–based enumeration scheme for solving time–constrained project scheduling problems with locally regular or locally concave objective functions can readily be generalized to the case where the limited capacity of renewable resources has to be taken into account. To this end we have to ensure that each partial schedule S^C considered in the course of the enumeration satisfies the renewable–resource constraints

$$\sum_{i\in C: S_i \leq t < S_i + p_i} r_{ik} \leq R_k \quad (k \in \mathcal{R}^\rho)$$

This can be achieved by deleting, for each activity $j^* \in V \setminus C$ not yet scheduled, all those tentative start times t from decision set $\mathcal{D}_{j^*}$ for which the scheduling of activity j^* at time t would lead to a resource conflict, i.e., for which

$$\sum_{i \in C: S_i \leq t' < S_i + p_i} r_{ik} + r_{j^*k} > R_k \text{ for some } k \in \mathcal{R}^\rho \text{ and some } t' \in [t, t + p_{j^*}[$$

Similarly to the relaxation–based enumeration scheme from Subsection 15.5.1.1, it may now happen that we reach a deadlock where we have to perform backtracking before having obtained a feasible schedule. This occurs every time when the current partial schedule S^C cannot be expanded further because the decision set $\mathcal{D}_{j^*}$ becomes void for some activity $j^* \in V \setminus C$ not yet scheduled. In this case, the earliest start time $t \geq ES_{j^*}$ of activity j^* satisfying the resource constraints is greater than latest start time LS_{j^*}.

15.5.2 Allocatable and synchronizing resources

In this subsection, we deal with allocatable and synchronizing resources, two new resource types that appear in a project scheduling application from service operations management to be discussed in more detail in Subsection 15.6.1. Originally, those resource types have been introduced by Schwindt and Trautmann (2003) in the context of production scheduling in the metal casting industry.

An *allocatable resource* is a special renewable resource k for which the r_{ik} resource units processing an activity i have to be allocated by some activity $a_k(i)$ starting no later than i. The r_{ik} resource units remain occupied from the start of allocating activity $a_k(i)$ until the completion of activity i. For example, such an allocatable resource k may correspond to equipment that has to be installed each time before being used for executing some activity i. If setting up the r_{ik} units needed also requires some scarce renewable resource k' like personnel or tools, the setup can be modeled as an activity $i' = a_k(i)$ with positive requirement $r_{i'k'}$ allocating the r_{ik} units of resource k. In this way we ensure that the units installed for processing activity i are not used by different activities j before i has been completed and releases the units.

Let $\mathcal{R}^\alpha$ be the set of all allocatable resources and let $V_k := \{i \in V \mid r_{ik} > 0\}$ denote the set of all activities using resource k. We say that a schedule S is *allocation–feasible* if at any point in time t no more than R_k units of a resource $k \in \mathcal{R}^\alpha$ have been allocated. By $\mathcal{A}_k(S,t) := \{i \in V_k \mid S_{a_k(i)} \leq t < S_i + p_i\}$ we denote the set of activities i for which at time t the units of resource k have been allocated. The resource constraints then read

$$\sum_{i \in \mathcal{A}_k(S,t)} r_{ik} \leq R_k \quad (k \in \mathcal{R}^\alpha,\ 0 \leq t \leq \bar{d})$$

Now assume that in an iteration of the relaxation–based enumeration scheme, we have obtained a schedule S causing a resource conflict on an allocatable resource k at some time t. We then resolve the conflict by determining a minimal delaying alternative B for set $\mathcal{A}_k(S,t)$ and introducing the precedence constraints

$$S_{a_k(j)} - S_i \geq p_i \quad (j \in B)$$

between some activity i from set $A = \mathcal{A}_k(S,t) \setminus B$ and the allocating activities $a_k(j)$ of all activities j from set B.

In certain applications of project scheduling, the parallel execution of multiple activities on a renewable resource requires that the activities are started at the same time. We then speak of the renewable resource as a *synchronizing resource.* As an example consider staff training in a class room, where people from different departments are trained on computers. If the limited class room capacity does not allow for training all departments simultaneously, we may represent the teaching of each department as a single activity. Since the training activities of all departments following the same course must start jointly, the class room is modeled as a synchronizing resource.

Let $\mathcal{R}^\sigma$ be the set of all synchronizing resources. Aside from the ordinary renewable–resource constraints (15.2), synchronizing resources give rise to so–called *simultaneity constraints*

$$S_i = S_j \quad (i,j \in V_k \cap \mathcal{A}(S,t) \text{ for some } k \in \mathcal{R}^\sigma \text{ and some } t \in [0,\bar{d}]) \quad (15.6)$$

requiring that the parallel execution of activities on a synchronizing resource must start at the same time. A schedule S satisfying constraints (15.2) and (15.6) is called *synchronization–feasible.*

Next, we show how to remove violations of the simultaneity constraints occurring for some schedule S. Assume that at time t the overlapping activities $i \in V_k \cap \mathcal{A}(S,t)$ are not started jointly. Constraints (15.6) say that for any two activities $i,j \in V_k$ we must ensure that $S_i = S_j$, $S_j \geq S_i + p_i$, or $S_i \geq S_j + p_j$. This can be achieved as follows. We select an activity $i \in V_k \cap \mathcal{A}(S,t)$, which is called the *synchronizing activity.* For the activities j overlapping with i on resource k but starting before i, we introduce the temporal constraints

$$S_j - S_i \geq 0 \quad (j \in V_k \cap \mathcal{A}(S,t) : S_j < S_i)$$

delaying the start of j to time S_i, and for the activities j overlapping with i on k but starting after i, we add the precedence constraints

$$S_j - S_i \geq p_i \quad (j \in V_k \cap \mathcal{A}(S,t) : S_j > S_i)$$

to prevent the parallel execution of i and j. Note that since inequality $S_j - S_i \geq p_i$ implies inequality $S_j - S_i \geq 0$, it is not necessary to consider, for fixed i, the case where precedence constraints $S_j \geq S_i + p_i$ are introduced for activities j being started earlier than synchronizing activity i. The case where for some activity j being started after i, we want to introduce constraint $S_i \geq S_j$ is obtained by choosing j to be the synchronizing activity.

15.5.3 Changeover resources

This section is concerned with renewable resources k whose units must be changed over when passing from one activity i to another activity j. We assume that the times ϑ_{ij}^k needed for the changeovers may depend on both activities i and j. This situation arises, for example, when several subprojects using common renewable resources are performed simultaneously at different sites. When a unit of resource k passes from the execution of activity i at location $\ell(i)$ to activity j carried out at a different location $\ell(j)$, the unit has to be torn down after the completion of i, transported from $\ell(i)$ to $\ell(j)$, and put into service for processing j. Hence, the total changeover time of resource k between the execution of activities i and j includes a sequence–dependent transportation time t_{ij} from $\ell(i)$ to $\ell(j)$.

We discuss an approach to project scheduling with changeover resources devised by Neumann et al (2003a, Section 2.14), following the presentation of Schwindt (2005, Section 5.2). Let $\mathcal{R}^\vartheta$ be the set of all renewable resources with changeovers, which are called *changeover resources*. For what follows we suppose that the weak triangle inequality

$$\vartheta_{hi}^k + p_i + \vartheta_{ij}^k \geq \vartheta_{hj}^k$$

is satisfied for all $k \in \mathcal{R}^\vartheta$ and all $h, i, j \in V_k$. This assumption is generally met in practice because otherwise it would be possible to save changeover time by processing additional activities. For notational convenience we additionally assume that there are neither changeovers from the project beginning 0 to activities $i \in V_k$ nor changeovers from activities $i \in V_k$ to the project termination $n+1$.

To meet the resource constraints, the demand for a changeover resource by activities and changeovers must not exceed the respective resource capacities at any point in time. More precisely, let for given changeover resource k, $X_k : V_k \rightarrow 2^{\mathbb{N}}$ be a mapping providing for each activity $i \in V_k$ the set of units of resource k processing activity i, i.e.,

$$|X_k(i)| = r_{ik} \quad (i \in V) \tag{15.7}$$

We call a schedule S *changeover–feasible* if for each resource $k \in \mathcal{R}^\vartheta$, mapping X_k can be chosen such that

$$\left.\begin{array}{r} S_j \geq S_i + p_i + \vartheta_{ij}^k \\ \text{or } S_i \geq S_j + p_j + \vartheta_{ji}^k \end{array}\right\} \quad (i, j \in V_k : i \neq j,\ X_k(i) \cap X_k(j) \neq \emptyset) \tag{15.8}$$

and

$$X_k(i) \subseteq \{1, \ldots, R_k\} \quad (i \in V_k) \tag{15.9}$$

Constraints (15.8) say that activities i and j (including the possible changeover in between) must not overlap if there is a unit of resource k processing both activities i and j. Condition (15.9) limits the availability of resource k to R_k units. Since all changeover times are nonnegative, a changeover–feasible schedule always observes the renewable–resource constraints (15.2).

In the following, we develop an equivalent characterization of the changeover–feasibility of schedules, which will serve as a basis for detecting and resolving resource conflicts. The analogue to schedule–induced strict order $O(S)$ introduced in Subsection 15.3.1 is the strict order

$$O^k(S) := \{(i,j) \in V_k \times V_k \mid i \neq j,\ S_j \geq S_i + p_i + \vartheta_{ij}^k\}$$

Let for given schedule S and resource $k \in \mathcal{R}^\vartheta$, X_k be a mapping satisfying conditions (15.7) and (15.8) and let $r_k(S) := |\bigcup_{i \in V_k} X_k(i)|$ denote the number of resource units used. Clearly, S is changeover–feasible exactly if $r_k(S) \leq R_k$ for all $k \in \mathcal{R}^\vartheta$. We consider an *antichain* U in schedule–induced strict order $O^k(S)$, i.e., a set of activities U such that for any two activities $i, j \in U$, neither $(i,j) \in O^k(S)$ nor $(j,i) \in O^k(S)$. It follows from the definition of $O^k(S)$ that $[S_i, S_i + p_i + \vartheta_{ij}^k[\cap [S_j, S_j + p_j + \vartheta_{ji}^k[\neq \emptyset$ for any two activities $i, j \in U$. Condition (15.8) then implies that $X_k(i) \cap X_k(j) = \emptyset$ for any $i, j \in U$. Exploiting equation (15.7) this means that $|\bigcup_{i \in U} X_k(i)| = \sum_{i \in U} |X_k(i)| = \sum_{i \in U} r_{ik}$. On the other hand, it is obvious that for any subset $U' \subseteq V_k$, the number $|\bigcup_{i \in U'} X_k(i)|$ of resource units occupied by activities from U' is less than or equal to the joint requirements $\sum_{i \in U'} r_{ik}$ for resource k. Consequently, $r_k(S)$ equals the weight $\sum_{i \in U_k} r_{ik}$ of a maximum–weight antichain U_k in $O^k(S)$. Since all activities from set U_k pairwise overlap in time, U_k can be regarded as an *active set* $\mathcal{A}_k(S)$ for S. Schedule S is changeover–feasible precisely if none of the active sets $\mathcal{A}_k(S)$ belonging to a changeover resource $k \in \mathcal{R}^\vartheta$ is forbidden.

A maximum–weight antichain U_k can be determined efficiently by computing a minimum $(0, n+1)$–flow x^k of value $v^k(x^k) = r_k(S)$ in the flow network $\overline{G}_k(O^k(S))$ with node set $V_k \cup \{0, n+1\}$ and arc set $\{\langle i,j \rangle \mid (i,j) \in O^k(S) \cup (\{0\} \times V_k) \cup (V_k \times \{n+1\})\}$, where nodes $i \in V_k$ are associated with lower capacities r_{ik}. Recall that we have assumed that $\vartheta_{0i}^k = \vartheta_{i,n+1}^k = 0$ for all $i \in V_k$. As a consequence, arcs $\langle 0, i \rangle$ and $\langle i, n+1 \rangle$ linking the source 0 and the sink $n+1$ to nodes $i \in V_k$ correspond to precedence relationships $S_i \geq S_0 + \vartheta_{i0}^k$ and $S_{n+1} \geq S_i + p_i + \vartheta_{i,n+1}^k$ that are established by schedule S, analogously to the remaining arcs $\langle i,j \rangle$ with $(i,j) \in O^k(S)$.

Computing the minimum $(0, n+1)$–flows x^k provides the answer to the question whether or not given schedule S is changeover–feasible. In the latter case, however, the question arises how to compute a forbidden active set $\mathcal{A}_k(S)$ for resolving the resource conflict. In contrast to the resource types considered so far, the active set depends on the assignment of activities i to resource units from $X_k(i)$. This problem can be solved by exploiting the duality relationship between minimum flows and maximum cuts in networks. At first, we notice that the lower node capacities r_{ik} can be transformed into equivalent arc capacities by splitting up every node $i \in V_k$ into two nodes i' and i'' linked by arc $\langle i', i'' \rangle$ with lower capacity $l_{i'i''} = r_{ik}$ and infinite upper capacity. In general, the network flow methods do not only provide a minimum $(0, n+1)$–flow x^k in $\overline{G}_k(O^k(S))$ but also a maximum $(0, n+1)$–cut $[U'_k, U''_k]$, whose capacity equals the minimum flow value $v^k(x^k)$. In addition, it can easily be shown that any maximum $(0, n+1)$–cut in $\overline{G}_k(O^k(S))$ only contains forward arcs. Thus, active set $\mathcal{A}_k(S)$ coincides with the set $\{i \in V_k \mid \langle i', i'' \rangle \in [U'_k, U''_k]\}$ of activities i for which the arc linking the two nodes i' and i'' is contained in cut $[U'_k, U''_k]$ (or, figuratively speaking, the set of all activities whose nodes are broken up by $[U'_k, U''_k]$).

Now assume that we have determined some forbidden active set $\mathcal{A}_k(S)$ for schedule S. To remove the resource conflict, we then compute a minimal delaying alternative B for $\mathcal{A}_k(S)$, choose an activity i from set $A = \mathcal{A}_k(S) \setminus B$, and add the temporal constraints

$$S_j - S_i \geq p_i + \vartheta^k_{ij} \quad (j \in B)$$

ensuring that each activity $j \in B$ is only started when the changeover from i to j has been completed.

15.5.4 Cumulative resources

Until now we have dealt with project scheduling problems where activities use renewable resources such as manpower or equipment, whose availability is independent of their previous utilization. In practice, we may often encounter additional scarce resources such as investment capital, storage space, or intermediate products, which are generally depleted and replenished over the execution time of the project. The availability of such *cumulative resources* at a given time t results cumulatively from all positive and negative requirements (depletions and replenishments, respectively) that have occurred by time t. Since the availability of a cumulative resource k can be regarded as the inventory level in some storage facility, we may also speak of a *reservoir* or a *storage resource* (see Laborie 2003 and Neumann et al 2003a, Section 2.12). The inventory level in resource k is supposed to be bounded from below by $\underline{R}_k \in \mathbb{Z}_{\geq 0}$ (e.g., a safety stock) and from above by $\overline{R}_k \in \mathbb{Z}_{\geq 0}$ with $\overline{R}_k \geq \underline{R}_k$ (e.g., the capacity of the storage facility).

In this paper, we restrict ourselves to *discrete* cumulative resources, which are depleted and replenished at discrete points in time like start or completion times of activities. The more general case of *continuous* cumulative resources, where the depletion and replenishment rates may be finite, is treated in Neumann et al (2003a, Subsection 2.12.2) and Neumann et al (2005).

Let $\mathcal{R}^\gamma$ be the set of all cumulative resources and let $r_{ik} \in \mathbb{Z}$ denote the demand of activity i for resource k. If $r_{ik} > 0$, activity i replenishes resource k by r_{ik} units, and if $r_{ik} < 0$, resource k is depleted by $-r_{ik}$ units. We assume that resources $k \in \mathcal{R}^\gamma$ are depleted at start times and replenished at completion times of activities. To simplify writing, we also suppose that an activity cannot deplete and replenish one and the same cumulative resource. r_{0k} corresponds to the initial stock of resource k.

Now let $V_k^- := \{i \in V \mid r_{ik} < 0\}$ and $V_k^+ := \{i \in V \mid r_{ik} > 0\}$ denote the sets of all activities $i \in V$ depleting and replenishing, respectively, resource k. For given schedule S

$$\mathcal{A}_k(S,t) := \{i \in V_k^- \mid S_i \leq t\} \cup \{i \in V_k^+ \mid S_i + p_i \leq t\}$$

is the *active set* of all activities that determine the inventory level

$$r_k(S,t) := \sum_{i \in \mathcal{A}_k(S,t)} r_{ik}$$

in resource k at time t. The *inventory constraints* can now be written as

$$\underline{R}_k \leq r_k(S,t) \leq \overline{R}_k \quad (k \in \mathcal{R}^\gamma,\ 0 \leq t \leq \overline{d}) \tag{15.10}$$

A schedule S which satisfies the inventory constraints (15.10) is termed *inventory–feasible.*

For given schedule S, we distinguish between two types of resource conflicts and forbidden sets. We speak of an *inventory shortage* at time t if the inventory in some resource k falls below the minimum inventory level. The active set $\mathcal{A}_k(S,t)$ then constitutes a so–called *k–shortage set* F with

$$\sum_{i \in F} r_{ik} < \underline{R}_k$$

Conversely, we may have an *inventory excess* where the inventory in some resource k exceeds the maximum inventory level. In this case, $\mathcal{A}_k(S,t)$ is termed a *k–surplus set*, i.e., a set F with

$$\sum_{i \in F} r_{ik} > \overline{R}_k$$

Similarly to the approach from Subsection 15.5.1.1, we introduce the concept of minimal forbidden sets, which will allow us to define appropriate minimal delaying alternatives for resolving the resource conflicts. In contrast to the case of renewable resources, our minimality concept cannot refer to inclusion–minimality because resource demands may be positive or negative. That is why we define a k–shortage set F to be a *minimal k–shortage set* if no depleting activity can be removed from F and no replenishing activity can be added to F without loosing the shortage property. Symmetrically, we say that a k–surplus set F is a *minimal k–surplus set* if deleting a replenishing or adding a depleting activity removes the inventory excess. As has been shown by Neumann and Schwindt (2002), a schedule S is inventory–feasible if and only if (i) each minimal k–shortage set F contains a replenishing activity $i \in V_k^+$ and a depleting activity $j \in V_k^-$ such that activity j is not started before activity i has been completed (i.e., $S_j \geq S_i + p_i$) and (ii) each minimal k–surplus set F contains a depleting activity $i \in V_k^+$ and a replenishing activity $j \in V_k^-$ such that activity j is not completed before activity i has been started (i.e., $S_j + p_j \geq S_i$). A minimal forbidden set F satisfying condition (i) or (ii) is said to be broken up in schedule S.

Now consider an inventory shortage for some resource $k \in \mathcal{R}^\gamma$ at time t. A minimal delaying alternative B for active k–shortage set $\mathcal{A}_k(S,t)$ is an inclusion–minimal subset of $\mathcal{A}_k(S,t)$ containing a depleting activity j of each minimal k–shortage set F that can be obtained from $\mathcal{A}_k(S,t)$ by deleting depleting and adding replenishing activities. By introducing the precedence constraints

$$S_j - S_i \geq p_i \quad (j \in B)$$

between some activity $i \in A = V_k^+ \setminus \mathcal{A}_k(S,t)$ (i.e., a replenishing activity i being completed after time t) and all activities $j \in B$, we break up all those of the above minimal k–shortage sets F that contain activity i.

The case of an inventory excess can be dealt with similarly. A minimal delaying alternative B now contains a replenishing activity j of each minimal k–surplus set F including all depleting activities of $\mathcal{A}_k(S,t)$ and no additional replenishing activities. The resource conflict at time t is removed by adding the temporal constraints

$$S_j - S_i \geq -p_j \quad (j \in B)$$

between some activity $i \in A = V_k^- \setminus \mathcal{A}_k(S,t)$ (i.e., a depleting activity i being started after time t) and all activities $j \in B$. We notice that those temporal constraints correspond to *maximum* time lags $d_{ji}^{max} = p_j$ between activities j and activity i. This observation provides some insight into the fact that the problem of finding a time– and inventory–feasible schedule is NP–hard even if project network N does not contain cycles (see Neumann and Schwindt 2002).

15.6 New applications

In this section, we briefly discuss two practical applications of the models and methods for resource–constrained project scheduling discussed thus far. The first application, which has been presented by Mellentien et al (2004), is located in the area of service operations management and deals with scheduling industrial event–marketing activities involving customers. The second application, where we are concerned with production planning in the process industries, will show that project scheduling techniques may also be successfully applied to problems that, strictly speaking, do not belong to project management. This application has been discussed by Schwindt and Trautmann (2000) for the first time.

15.6.1 Factory pick–up of new cars

Car manufacturers increasingly organize visit programs for the customers that pick up their new cars at the factory. Such a program consists of a broad range of event–marketing activities and is designed to establish an emotional relationship between the customer and the brand. We study the problem of scheduling all program activities of one day in such a way that the sum of the customers' waiting times during their visit is minimized. In service operations management, short customer waiting times are considered to be a key performance indicator of customer satisfaction (see e.g., Fitzsimmons and Fitzsimmons 2004, Chapter 11).

Let C be the set of all customers visiting the factory at a given day. We assume that the visitors perform their visit programs in groups. All visitors of a group participate simultaneously in the program items chosen. It may happen that a visitor does not select the full program of the group. In this case, he or she will leave earlier if all program items chosen have been run. Otherwise, he or she will have to wait for the start of the next program item. For each group of visitors, we introduce one activity per program item that has been chosen by at least one customer of the group. By V_c we denote the set of all activities performed by customer $c \in C$. For given schedule S, the total waiting time of all customers then equals

$$\sum_{c \in C}(\max_{i \in V_c}(S_i + p_i) - \min_{i \in V_c} S_i - \sum_{i \in V_c} p_i)$$

If we consider fictitious start and completion activities $\alpha(c)$ and $\omega(c)$ of duration zero for each customer c and subtract the constant sum $\sum_{c \in C} \sum_{i \in V_c} p_i$ of activity durations, the objective function can be written as

$$f(S) = \sum_{i \in C}(S_{\omega(c)} - S_{\alpha(c)})$$

and represents a sum of weighted start times, i.e., a linear function $\sum_{i \in V} v_i S_i$ with $v_i = 1$ or $v_i = -1$ if $i = \omega(c)$ or $i = \alpha(c)$, respectively, for some $c \in C$, and with $v_i = 0$, otherwise.

Organizational requirements like the necessity to welcome a customer before starting any other program item give rise to precedence constraints between certain activities $i \in V_c$ in the program of a customer $c \in C$. To ensure that $\alpha(c)$ and $\omega(c)$ represent the start and completion, respectively, of customer c's program, we also introduce the minimum time lags $d^{min}_{\alpha(c)i} = 0$ and $d^{min}_{i\omega(c)} = p_i$ for all activities $i \in V_c$. Finally, we have to ensure that all customer programs are completed within one working day. This requirement is modeled by putting the maximum project duration $d^{max}_{0,n+1}$ to be equal to the length $\overline{d}$ of one working day and linking the project beginning 0 and the individual start activities $\alpha(c)$ by minimum time lags $d^{min}_{0,\alpha(c)} = 0$ and the individual completion activities $\omega(c)$ with the project completion $n + 1$ by minimum time lags $d^{min}_{\omega(c),n+1} = 0$.

For executing a program item, some staff like driving instructors and different kinds of facilities such as handover bays or a cinema are needed. A staff member can perform only one program item at a time. Moreover, according to the "one face to the customer" principle, all activities of a group must be supervised by a customer advisor, who guides the group through the entire program. Certain facilities can be used by a given number of groups at the same time, whose activities then sometimes need to be synchronized (e.g., during a cinema show).

Basically, each type of facilities and each staff group is modeled as a renewable resource k whose capacity is shared among the activities of different groups. With respect to the temporal constraints, a customer may attend certain program items simultaneously. To prevent the overlapping execution of items being performed by one and the same customer, we introduce an extra renewable resource with capacity one for each group and assign a resource requirement of one to each of the group's activities. In this way, we ensure that any feasible schedule arranges the program items of a customer into some sequence, possibly with waiting times in between.

The concept of allocatable resources allows us to integrate the "one face to the customer" principle into our model. As a consequence of this principle, a group must constantly be accompanied by the same customer advisor. Accordingly, the customer advisors form an allocatable resource k. One unit of this resource is to be allocated at the start of the initial activity of each group and released after the group has performed all program items selected. This can be achieved by choosing, for each group, allocating activity $a_k(i)$ to a fictitious program start activity and releasing activity i to a fictitious program end activity of the group. In this way, the advisor remains allocated to the group during the entire program.

Finally, we use a synchronizing resource k for modeling the requirement that certain activities have to be executed jointly for different groups. We consider the example of an advertising movie that is shown in a factory cinema. Of course, for all groups watching the movie in parallel, the projection starts at the same time, or, to put it differently, the movie activities have to be synchronized on resource k. The capacity R_k of resource k equals the number of seats in the cinema. The number r_{ik} of units taken up by the movie activity i of a group corresponds to the number of group members that have selected this item in their program.

Table 15.1 recapitulates the different features of the application and their counterparts in the project scheduling model.

Table 15.1. Factory pick–up of new cars: application and model

Scheduling of customer visit programs	Project scheduling
Program items of a group	Activities
Customers' waiting times	Sum of weighted start times
Organizational requirements	Temporal constraints
Consecutive execution of activities	Single-unit renewable resources
Facilities and staff	General renewable resources
"One face to the customer" principle	Allocatable resources
Joint execution of activities	Synchronizing resources

15.6.2 Batch scheduling

In the process industries, final products arise from several successive chemical or physical transformations of bulk goods, liquids, or gases processed on *processing units* such as reactors, heaters, or filters. The transformation of input products into output products on a given processing unit is called a *task*. Each task may consume several input products and may produce several output products, whose amounts may be chosen within prescribed bounds. Perishable products must be consumed within a given shelf life time, which may be equal to zero. In addition, the storable intermediate products must be stocked in dedicated *storage facilities* like tanks or silos. Further peculiarities encountered in the process industries are cyclic product structures and sequence–dependent cleaning times on processing units.

The following material draws from Schwindt (2005, Section 6.3). Throughout this subsection we assume that the production is operated in *batch mode*, which means that at the beginning of a task, the input products are loaded into the processing unit, and the output becomes available at the termination of the task. The combination of a task and the corresponding quantity produced is called a *batch*. An *operation* corresponds to the processing of a batch. Since the batch sizes are limited by the capacity of the processing units, a task is generally executed more than once, resulting in several corresponding operations.

In contrast to manufacturing, the processing times of operations are mostly independent of the respective batch sizes.

The short–term production scheduling problem considered in what follows consists in allocating processing units and storage facilities over time to the production of given primary requirements such that all operations are completed within a minimum makespan. This objective is particularly important in batch production, where often a large number of different products are processed on multi–purpose equipment. In this case, the production plant is configured according to the set of production orders released. Before processing the next set of production orders, the plant generally has to be rearranged, which requires the completion of all operations.

In general, this production scheduling problem is modeled as a large–scale mixed–integer linear program based on a discrete–time or a continuous–time approach (see Floudas and Lin 2004 for a review of those two model types). The special feature of the heuristic method by Schwindt and Trautmann (2000) is the decomposition of the production scheduling problem into a batching problem and a batch scheduling problem. The batching phase generates appropriate batches, which in the course of the batch scheduling phase are scheduled on the processing units subject to inventory constraints. Neumann et al (2002a) have shown that the batching problem can be formulated as a mixed–binary linear program. As will be explained in what follows, the batch scheduling problem can be modeled as a resource–constrained project scheduling problem with changeover and cumulative resources.

Assume that we have obtained a feasible solution to the batching problem that provides us with a set of operations to be scheduled on the processing units. The batch scheduling problem consists in allocating the resources to the operations over time such that the processing of all batches is completed within a minimum amount of time. A variety of technological and organizational constraints have to be taken into account. A task generally requires different types of resources: processing units with sequence-dependent cleaning times, input products, and storage facilities for output products. The availability of these resources is limited by capacities and inventories. Finally, there may exist perishable intermediate products.

The execution of all operations can be viewed as a project, where the makespan to be minimized corresponds to the project duration S_{n+1}. Accordingly, we identify each operation i with one real activity $i \in V$, with duration p_i being equal to the processing time of the corresponding task. Each operation is executed on a processing unit. We combine identical processing units to form a pool. Each pool is modeled as a changeover resource $k \in \mathcal{R}^{\vartheta}$. The requirement r_{ik} of activity i for resource k equals 1 if operation i is carried out on a processing unit of the pool, and 0, otherwise. The capacity R_k of resource k is equal to the number of processing units in the corresponding pool. The cleaning times between consecutive operations on a processing unit can be interpreted as sequence–dependent changeover times between the activities. The changeover time ϑ_{ij}^k between two activities i and j on resource k equals the cleaning time after operation i if passing from i to j requires a cleaning of resource k.

Intermediate storage facilities can be modeled as cumulative resources. We associate one cumulative resource $k \in \mathcal{R}^\gamma$ with each intermediate product to be stocked. The minimum inventory level $\underline{R}_k$ of resource k equals zero, and the maximum inventory level $\overline{R}_k$ is set to be equal to the capacity of the storage facility for the product. The demands r_{ik} of activities i for resource k can be determined as follows. If the product is an input product of operation i, r_{ik} equals the negative amount of the product consumed by i, and if the product is an output product of operation i, r_{ik} equals the amount of the product produced by i.

Finally, we turn to perishable intermediate products. We only consider the case of chemically unstable products, which must be consumed immediately. The case of general shelf life times can be modeled by introducing auxiliary activities and cumulative resources (see Neumann et al 2005). The immediate consumption of an intermediate product can be enforced by associating the product with a cumulative resource k whose maximum inventory level is equal to zero and modeling the resource demands as described above for the storable intermediate products.

In conclusion, Table 15.2 summarizes the modeling of the batch scheduling problem as a resource–constrained project scheduling problem.

Table 15.2. Batch scheduling: application and model

Batch scheduling	Project scheduling
Operations	Activities
Makespan	Project duration
Pools of identical processing units	Changeover resources
Cleaning times	Sequence-dependent changeover times
Intermediate storage facilities and perishable intermediate products	Cumulative resources

15.7 Conclusions

In this chapter, we have discussed recent results on deterministic resource–constrained project scheduling with time windows. We have presented an order–based structural analysis of the feasible region of project scheduling problems and a new classification of objective functions important to practice. This structural analysis and classification have been exploited for developing efficient solution approaches. First, methods for solving time–constrained project scheduling problems have been proposed. Second, the resolution of conflicts for renewable, allocatable, synchronizing, changeover, and cumulative (or storage) resources and thus the solving of corresponding resource–constrained project scheduling problems have been studied. Finally, we have briefly discussed two applications of resource–constrained project scheduling: factory pick–up of new cars and batch scheduling in process industries.

An important area of future research is the development of efficient *decomposition methods* for solving large–scale project scheduling problems, e.g., batch scheduling in process industries with several thousand operations. Only a few approaches have recently been proposed in that area (see Gentner et al 2004 and Gentner 2005).

Another field of future research is *rescheduling*, where due to disturbances in practice, a feasible baseline schedule has become infeasible and must be repaired (cf. Subsection 15.3.2). Instead of or in addition to repairing baseline schedules, *robust schedules* may be constructed, e.g., by inserting different kinds of appropriate buffers. An overview of some first results on how to deal with rescheduling and build robust schedules can be found in Demeulemeester and Herroelen (2002, Chapter 10) and Schwindt (2005, Section 6.5), and a new approach has been presented by Mellentien (2005).

A third area of future research is resource–constrained *project scheduling with given work content*. Instead of prescribed processing times and resource requirements of activities, a work content (e.g., amount of man–days) may be given for each activity. In that case, the resource utilization of each activity may vary during its execution between certain limits. A first comprehensive approach to modeling and solving such project scheduling problems has been proposed by Fündeling (2005).

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B. (1993). *Network Flows*, Prentice Hall, Englewood Cliffs.

Bartusch, M., Möhring, R.H., and Radermacher, F.J. (1988). Scheduling project networks with resource constraints and time windows, *Annals of Operations Research* 16:201–240.

Demeulemeester, E.L., Herroelen, W.S. (2002). *Project Scheduling – A Research Handbook*, Kluwer, Boston.

De Reyck, B., Herroelen, W.S. (1998). A branch–and–bound procedure for the resource–constrained project scheduling problem with generalized precedence relations, *European Journal of Operational Research* 111:152–174.

Fitzsimmons, J.A., Fitzsimmons, M.J. (2004). *Service Management: Operations, Strategy, Information Technology*, McGraw–Hill, Boston.

Franck, B., Neumann, K., Schwindt, C. (2001). Truncated branch-and-bound, schedule–construction, and schedule–improvement procedures for resource–constrained project scheduling, *OR Spektrum* 23:297–324.

Floudas, C.A., Lin, X. (2004). Continuous–time versus discrete–time approaches for scheduling of chemical processes: A review, *Computers and Chemical Engineering* 28:2109–2129.

Fündeling, C.-U. (2005). Ressourcenbeschränkte Projektplanung bei gegebenen Arbeitsvolumina: Modellierung und Branch–and–Bound–Verfahren, *Report WIOR-662*, Institute for Economic Theory and Operations Research, University of Karlsruhe.

Gentner, K. (2005). *Dekompositionsverfahren für die ressourcenbeschränkte Projektplanung*, Shaker, Aachen.

Gentner, K., Neumann, K., Schwindt, C., Trautmann, N. (2004). Batch production scheduling in the process industries, in: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J.Y–T. Leung, Chapman Hall/CRC Press, Boca Raton, pp. 48.1–48.21.

Goldberg, A.V. (1997). An efficient implementation of a scaling minimum–cost flow algorithm, *Journal of Algorithms* 22:1–29.

Grötschel, M., Lovasz, L., Schrijver, A. (1995). *Geometric Algorithms and Combinatorial Optimization*, Springer, Berlin.

Herroelen, W.S., Van Dommelen, P., Demeulemeester, E.L. (1997). Project network models with discounted cash flows: A guided tour through recent developments, *European Journal of Operational Research* 100:97–121.

Icmeli, O., Erengüç, S.S. (1996). A branch and bound procedure for the resource–constrained project scheduling problem with discounted cash flows, *Management Science* 42:1395–1408.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results, *Artificial Intelligence* 143:151–188.

Mellentien, C. (2005). Präventive and reaktive Projektplanungsverfahren für ungewisse Projektumgebungen, *Report WIOR-663*, Institute for Economic Theory and Operations Research, University of Karlsruhe.

Mellentien, C., Schwindt, C., Trautmann, N. (2004). Scheduling the factory pick–up of new cars, *OR Spectrum* 26:579–601.

Neumann, K., Schwindt, C. (2002). Project scheduling with inventory constraints, *Mathematical Methods of Operations Research* 56:513–533.

Neumann, K., Zimmermann, J. (1999). Methods for resource–constrained project scheduling with regular and nonregular objective functions and schedule–dependent time windows, in: *Project Scheduling: Recent Models, Algorithms, and Applications*, J. Węglarz, ed, Kluwer, Boston, pp. 261–287.

Neumann, K., Zimmermann, J. (2000). Procedures for resource levelling and net present value problems in project scheduling with general temporal and resource constraints, *European Journal of Operational Research* 127:425–443.

Neumann, K., Zimmermann, J. (2002). Exact and truncated branch–and–bound procedures for resource–constrained project scheduling with discounted cash flows and general temporal constraints, *Central European Journal of Operations Research* 10:357–380.

Neumann, K., Nübel, H., Schwindt, C. (2000). Active and stable project scheduling, *Mathematical Methods of Operations Research* 52:441–465.

Neumann, K., Schwindt, C., Trautmann, N. (2002a). Advanced production scheduling for batch plants in process industries, *OR Spectrum* 24:251–279.

Neumann, K., Schwindt, C., Zimmermann, J. (2002b). Recent results on resource–constrained project scheduling with time windows: models, solution methods, and applications, *Central European Journal of Operations Research* 10:113–148.

Neumann, K., Schwindt, C., Zimmermann, J. (2003a). *Project Scheduling with Time Windows and Scarce Resources*, Springer, Berlin.

Neumann, K., Schwindt, C., Zimmermann, J. (2003b). Order–based neighborhoods for project scheduling with nonregular objective functions, *European Journal of Operational Research* 149:325–343.

Neumann, K., Schwindt, C., Trautmann, N. (2005). Scheduling of continuous and discontinuous material flows with intermediate storage restrictions, *European Journal of Operational Research* 165:495–509.

Nübel, H. (1999). A branch–and–bound procedure for the resource investment problem subject to temporal constraints, *Report WIOR–574*, Institute for Economic Theory and Operations Research, University of Karlsruhe.

Nübel, H. (2001). The resource renting problem subject to temporal constraints, *OR Spektrum* 23:359–381.

Russell, A.H. (1970). Cash flows in networks, *Management Science* 16:357–373.

Schwindt, C. (2000). Minimizing earliness–tardiness costs of resource–constrained projects, in: *Operations Research Proceedings 1999*, K. Inderfurth, G. Schwödiauer, W. Domschke, F. Juhnke, P. Kleinschmidt and G. Wäscher, eds, Springer, Berlin, pp. 402–407.

Schwindt, C. (2005). *Resource Allocation in Project Management.* Springer, Berlin.

Schwindt, C., Trautmann, N. (2000). Batch scheduling in process industries: An application of resource–constrained project scheduling, *OR Spektrum* 22:501–524.

Schwindt, C., Trautmann, N. (2003). Scheduling the production of rolling ingots: Industrial context, model, and solution method, *International Transactions in Operational Research* 10:547–563.

Schwindt, C., Zimmermann, J. (2001). A steepest ascent approach to maximizing the net present value of projects, *Mathematical Methods of Operations Research* 53:435–450.

Selle, T., Zimmermann, J. (2003). A bidirectional heuristic for maximizing the net present value of large-scale projects subject to limited resources, *Naval Research Logistics* 50:130–148.

Vanhoucke, M., Demeulemeester, E.L., Herroelen, W.S. (2001). An exact procedure for the resource–constrained weighted earliness–tardiness project scheduling problem, *Annals of Operations Research* 102:179–196.

Węglarz, J. (Ed.) (1999). *Project Scheduling: Recent Models, Algorithms, and Applications*, Kluwer, Boston.

Zimmermann, J. (2001). *Ablauforientiertes Projektmanagement: Modelle, Verfahren und Anwendungen*, Gabler, Wiesbaden.

Chapter 16

CP-BASED DECISION SUPPORT FOR PROJECT DRIVEN MANUFACTURING

Zbigniew A. Banaszak
Faculty of Electronics and Computer Science, Technical University of Koszalin, Sniadeckich 2,75-453 Koszalin, Poland
banaszak@tu.koszalin.pl

Abstract Some of the most challenging issues that arise in the domain of distributed manufacturing technology and management include manufacturability analysis, validation and evaluation of process plans, partnership in virtual enterprises, process design, and optimization of production plans and schedules. These issues are easily unified within a framework of a project-driven manufacturing concept which is focusing on small and medium size enterprises (SMEs) where products are manufactured based on make-to-order or build-to-order principle.

Regardless of character and scope of business activities a modern enterprise, has to build a project-driven development strategy in order to respond to challenges imposed by growing complexity and globalization. Managers need to be able to utilize a modern decision support tools as to undertake optimal business decisions in further strategic perspective of enterprise operation. In this context this contribution covers various issues of project management engineering while focusing in the field of Project-Driven Manufacturing, particularly in domains regarding the development of novel constraint programming based mathematical models and related decision-making methods as well as their implementation into the task oriented decision support systems aimed at project-driven SMEs management.

Keywords: Decision support, constraint logic programming, production planning, modeling, scheduling

16.1 Introduction

The key factor for companies confronting the challenge of remaining competitive in an era of globalization is undoubtedly the capability to fast and accurate decision making, especially in project management domain. Currently, the field of project-oriented management of manufacturing systems is driven primarily by market forces. Some of the most challenging issues that arise in the domain of distributed manufacturing technology and management include manufacturability analysis, validation and evaluation of process plans, partnership in extended enterprises, process design, and optimization of production plans and schedules. These issues are easily unified within a framework of a project-driven manufacturing concept which is focusing on small and medium size enterprises (SMEs) where products are manufactured based on make-to-order or build-to-order principle (see Kis et al (2004)).

In that context our objective is to provide a constraint programming based methodology aimed at designing of task oriented decision support systems (DSS) in particular oriented to the project management tasks in SMEs. Two purposes are considered. Firstly, to contribute to the problem of DSS designing by providing a new modeling framework unifying a customer-producer model and the state space pruning strategies as well as programming languages available. Secondly, to present the possible implementations of the programming methodology provided, i.e., showing a way enabling searching strategy evaluation, searching for adjustment of programming consistency (including problem statement and its specification, implemented constraint programming language, and the searching strategy applied), and showing an illustrative example of a software package application to a production order evaluation in the SME.

16.1.1 Multi-project environment

Finding an answer to the question whether a given production order can be accepted to be processed in a SME seems to be a fundamental from the customer-driven, and highly competitive market point of view. In order to decide whether a new production order (i.e., a new project) can be executed in a given production system, the producer capabilities and the customer needs have to be taken into account. So, the question facing a decision maker is whether the consumer's requirements can be balanced with the producer's capability (see Figure 16.1).

In that context decision making regards to the question whether enterprise's capability allows to fulfill constraints imposed by the production order requirements, i.e. whether its completion time, batch size, and its delivery period satisfy the customer requirements while satisfying constraints imposed by the enterprise configuration taking into account available resources, know how, experience, and so on. In the case of the response to this question being positive, i.e. there exist a way guaranteeing to complete a production order, the next question regards of finding of the most efficient one (e.g. as to be competitive on the market) (see Banaszak et al (2005), Banaszak and Zaremba (2004a), Banaszak (2003)).

The question stated above is usually considered under assumption that in the enterprise several project-like production orders are executed, however some production capabilities are still available. So, the question is whether the production order at hand can be accepted for execution in an enterprise where some resources availability is constrained in time?

The solution to such a general question faced by the decision maker will be either acceptance of the production order (based on production flow specifications, including manufacturing and transportation routes, batch sizes, schedules, etc.) or negotiation of a new production order requirements that must be satisfied in order to balance the enterprise capability with the production order completion constraints.

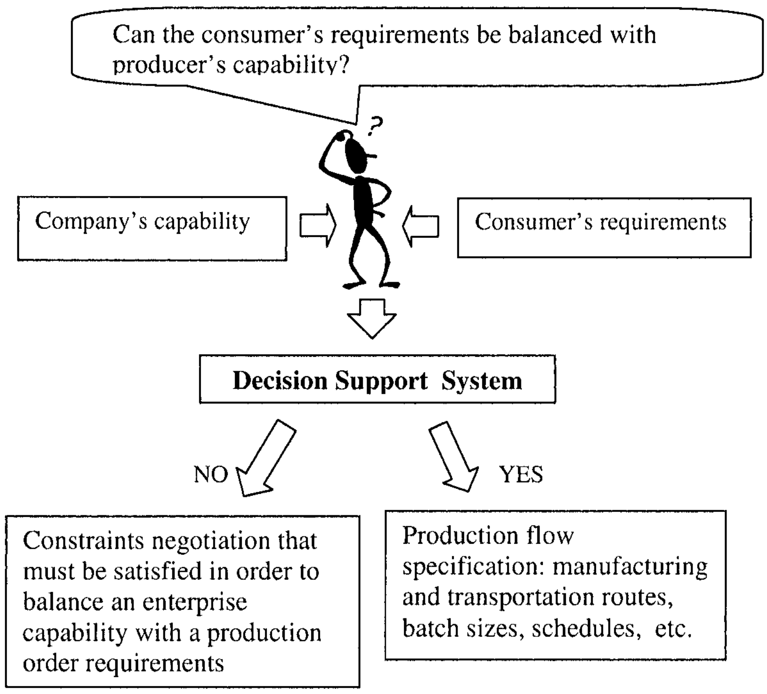

Can the consumer's requirements be balanced with producer's capability?
?
Company's capability
Consumer's requirements
Decision Support System
NO
YES
Constraints negotiation that must be satisfied in order to balance an enterprise capability with a production order requirements
Production flow specification: manufacturing and transportation routes, batch sizes, schedules, etc.

Figure 16.1. Decision making as a problem of balancing the consumer's requirements with the producer's capability.

Most companies, particularly SMEs have to manage various projects, which share a pool of constrained resources, taking into account various objectives at the same time. Due to the surveys [Lova, et al, 2000] conducted about 80% of companies have to deal with multiple projects, what corresponds to the other data stating that about 90% of all projects occur in the multi-project context.

Since tools supporting fast and cheap as well as accurate and on-line decision making are crucial to survive producers' competition, hence the methodology allowing to design task oriented (i.e., encompassing SMEs needs) decision support systems can be seen as one of among the most challenging issues.

16.1.2 Decision support

The multi-mode and resource-constrained models of the project-oriented management problems are mostly used to develop solution methods. It means, they employ the assumptions under which the solutions provided can be justified as an efficient, optimal, rational, and so on. The resource-constrained project scheduling problems (see e.g. Brucker et al (1999) for a survey) focus on the project makespan which has to be minimized. In turn, each activity of the multi-mode problem can be executed in one of several modes representing a relation between resource requirements of the activity and its duration. The schedule has to be precedence- and resource-feasible, and no activity may be interrupted.

The objective is to find an assignment of modes to activities as well as precedence- and resource-feasible starting times of all activities such that the makespan of the project is minimized. The problem is strongly NP-hard (see e.g. Blazewicz et al (1983)), so because of that commercially available software packages implementing the local search metaheuristics are quite costly and require skilful and well trained personnel.

So, new methods and techniques aimed at real-life constraints imposing on-line decision making are of great importance (see Anavi-Isakow and Golany (2003), Wei et al (2002)). They have to enhance an on-line project management, and support a manager in the process of decision making, e.g. in the course of an evaluation whether a new project can be accepted for processing in a multi-project environment of a manufacturing system at hand. They can also be included into DSS tools integrated into standard project management software like MS Project or CA-Super Project.

The problem has been addressed using two approaches: the scheduling of a single project and the scheduling of multiple simultaneous projects (see Brucker et al (1999), Anavi-Isakow and Golany (2003)). Most of the publications on project management have been dedicated to a single project. In recent years, however, there has been a growing interest in problems related to project scheduling in multi-project environments. In the single-project problem, satisfying time constraints is one of the dominant criteria. In contrast, scheduling of several projects with common and constrained resources takes into account other criteria such as idle resources, resource leveling, in-process inventory, and project splitting (see Lova et al (2000), Wei et al (2002)).

In this context it is worth to note that the currently available software tools allow pre-emption; however, they are not designed to cope with company production capability constraints in terms of resource and time availability, i.e., the schedules generated are usually infeasible if resource constraints are taken into account. Moreover, they do not permit to consider production planning in a unified way to enable an integrated approach to such different tasks as production and transportation routings, production and batch sizing as well as tasks scheduling.

Therefore, the objective is to find a computationally effective method (i.e., enabling decision maker to interact in an on-line mode) aimed at project-like production flow planning under constraints imposed by the multi-project environment. In other words, the method should be able to cope with the problem defined in terms of finding of a feasible schedule that satisfies the constraints imposed by the duration of production order processing, the cost assumed, and the time-constrained resources availability

16.1.3 Towards unified framework for dedicated applications

Regardless of its character and scope of business activities a modern enterprise, has to build a project-driven development strategy in order to respond to challenges imposed by growing complexity and globalization. Managers need to be able to utilize a modern DSS tools as to undertake optimal business decisions in further strategic perspective of enterprise operation.

Often repeating requests regard the questions such as: Whether in a given enterprise employed with the machine tools, automated guided vehicles (AGVs), buffers and warehouses a production order submitted can be completed due assumed period of time? Can the consumer's requirements regarding the final cost production be guaranteed? Does a given number of transportation means guarantee due time product delivery? Is the production capacity of the company sufficient to accept a new production order? Is the company able to respond? How to obtain such a response in an on-line mode? What strategy of production order processing is the most efficient one? Can the consumer's requirements be fulfilled within the assumed Extended Enterprise structure? Does the assumed set of SMEs guarantee a resultant Extended Enterprise to accomplish a given production order?

Respond to the questions usually involve many different aspects and contexts, e.g., money flow, personnel and/or resources allocation, tasks scheduling, workflows planning, and so on. The commercially available tools are just very task oriented, e.g., aimed at delivery planning, a layout designing, a product routing, etc.

Therefore a unified approach to such traditionally separately approached problems as layout designing, production and transportation routing, batch sizing, scheduling, and so on, enabling to integrate them into a DSS dedicated to the needs and requirements of particular enterprise plays a pivotal role in decision making. In other words, the approach required has to provide a unified framework allowing one to cope with different in scope and type decision problems within traditionally organized (see Banaszak and Zaremba (2004b) as well as extended enterprises (see Banaszak and Zaremba (2004b)).

In that context, the Constraint Programming/Constraint Logic Programming (CP/CLP) languages (see Wallace (2000), Rossi (2000)) by employing the constraints propagation concept and by providing unified constraints specification, can be considered as a well-suited framework for development of decision making software aimed at supporting the SMEs in the course of the Production Process Planning (PPP). Because of their declarative nature, for a user that is enough to state *what* has to be solved instead *how* to solve it (see Barták (2003), Barták (2004)) the approach seems to be very friendly for modeling of a company real-life and day-to-day decision-making (see Barták (1998), Barták (2003)).

16.2 Modelling framework

The objective concerns of computationally effective approach aimed at decision-making support for project-driven manufacturing in small- and medium-size enterprises. The problem considered regards of CP/CLP-based modeling framework enabling to provide an interactive scheduling of a new project subject to constraints imposed by a multi–project environment.

16.2.1 Customer–producer model

In order to balance the producer's capabilities with the customer's requirements a customer-consumer model is proposed. The model consists of a production system model, which reflects the parameters of a potential production system, and a production order model, which takes into account the order's requirements.

The model of a production system and the model of production order include parameters (such as: set of constraints, sets of discrete decision variables), which assure the correctness of obtained solutions and their application.

The discrete variables reflect various parameters – from the resource availability periods, through the production and transportation batch sizes, to the deadlines and taking over prices of the particular batches, Figure 16.2, and Figure 16.3.

Some of the variables may be grouped reflecting the traditional structure of a shop design, e.g., the transportation, and machining parts as well as may

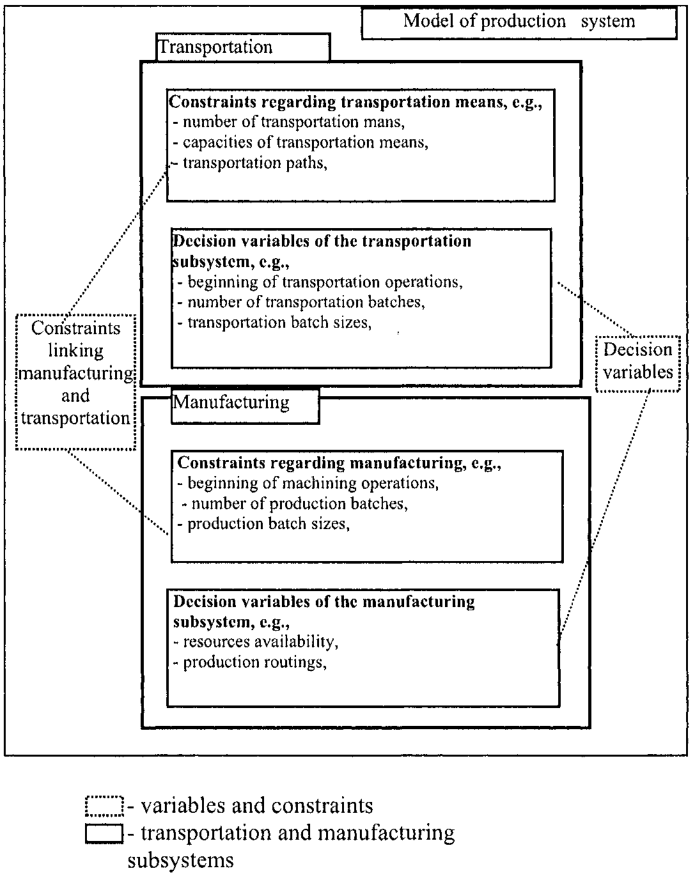

Model of production system
Transportation
Constraints regarding transportation means, e.g.,
- number of transportation mans,
- capacities of transportation means,
- transportation paths,
Decision variables of the transportation subsystem, e.g.,
- beginning of transportation operations,
- number of transportation batches,
- transportation batch sizes,
Constraints linking manufacturing and transportation
Decision variables
Manufacturing
Constraints regarding manufacturing, e.g.,
- beginning of machining operations,
- number of production batches,
- production batch sizes,
Decision variables of the manufacturing subsystem, e.g.,
- resources availability,
- production routings,
- variables and constraints
- transportation and manufacturing subsystems

Figure 16.2. Model of production system

be related each other, e.g., transportation and manufacturing batch sizes. The relations linking particular variables can be treated than as constraints determining the enterprise capabilities and/or production order requirements. So, the assumed model of the customer-producer consists of two standard elements: variables and their domains, constraints relating some subsets of variables. The model structure directly corresponds to so called constraint satisfaction problem. It should be noted, however, that the problems specified in the model are the decision making ones, i.e., problems concluding in questions awaiting either for "yes" or "not" response.

The above representation can be applied to the production flows observed in project-driven extended enterprises (see Banaszak and Zaremba (2004b), Banaszak and Pisz (2003), Banaszak et al (2003), Leach (2000)) as well as in

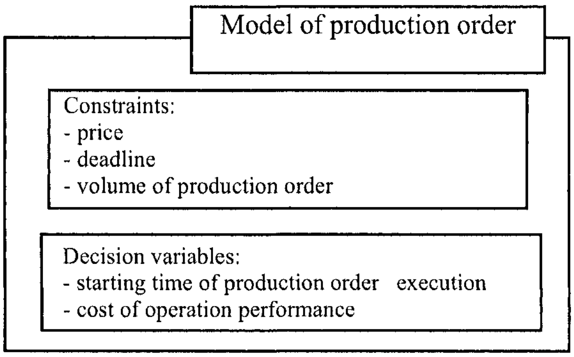

Model of production order
Constraints:
- price
- deadline
- volume of production order
Decision variables:
- starting time of production order execution
- cost of operation performance

Figure 16.3. Model of production order.

SMEs (see Lamma et al (1997), Anavi-Isakow and Golany (2003)). That is because the common platform provided by the workflow model (see Chin-Yin (2002)) describes the workflows of tasks, activities, etc., throughout the process of production flow planning (i.e., project management). The workflow model can be defined as a relationship among actions/activities treated as components of the considered processes.

16.2.2 Constraint Satisfaction Problem

The declarative character of Constraint Programming languages and a high efficiency in solving combinatorial problems creates an attractive alternative for the currently available (based on conventional operation research techniques) systems of computer-integrated management.

The Constraint Satisfaction Problem (CSP) consists of a set of variables $X = \{x_1, x_2, \ldots, x_n\}$, their domains $D = \{D_i | D_i = [d_{i1}, d_{i2}, \ldots, d_{ij}, \ldots, d_{im}], i = 1, ..., n\}$, and a set of constraints $C = \{C_i | i = 1, \ldots, L\}$. A solution is such an assignment of the variable values that all the constraints are satisfied.

It's easy to notice, that efficiency of a searching strategy can be evaluated in advance on the base of a given domains' sizes. In order to illustrate it let us consider $CSP = ((X, D), C)$, such that $X = \{A, B\}, D = \{D_A, D_B\}$, where $D_A = \{1, 2, 3\}$, $D_B = \{3, 4, \ldots, 9\}$, and $C = \{c_1, c_2\}$, where $c_1 = P[B \geq 3 \cdot A]$, and $c_2 = P[A + B > 9]$. Depending on the order in which variables are distributed the time required to obtain a set of feasible solutions may differ dramatically. In the case considered, starting with variable A requires twice less searching than in the case when variables distribution begin from variable B, see Figure 16.4.

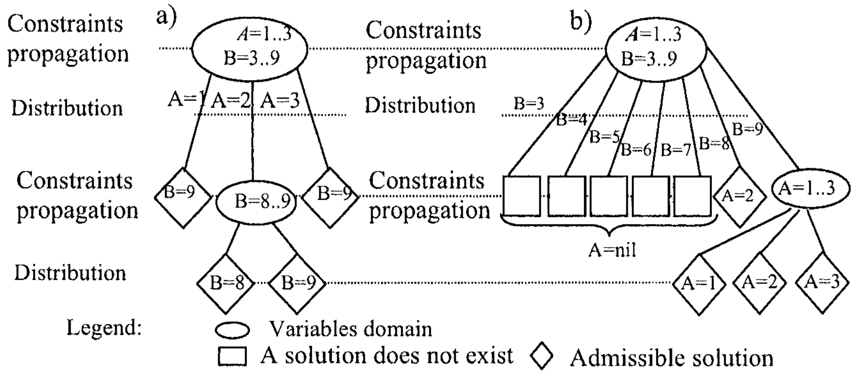


Figure 16.4. Decision variables distribution a) B follows A, b) A follows B.

For discussion let us consider the $CSP = ((X, D), C)$, where $X = \{x_1, x_2, \ldots, x_{12}\}$, $D = \{D_1, D_2, \ldots, D_{12}\}$, $C = \{c_1, c_2, \ldots, c_8\}$, and where: $c_1 = P[x_1, x_2, x_3]$, $c_2 = P[x_2, x_4, x_5]$, $c_3 = P[x_4, x_6]$, $c_4 = P[x_7, x_8]$, $c_5 = P[x_4, x_7]$, $c_6 = P[x_9, x_{10}]$, $c_7 = P[x_8, x_9]$, $c_8 = P[x_{11}, x_{12}]$.

The problem in natural way decomposes into subproblems, in particular into elementary problems, which are not further decomposed. The elementary problems can be seen as problems encompassed by constraints, for instance the elementary problem associated to the constraint $c_8 = P[x_{11}, x_{12}]$ can be stated as follows $CSP_8 = ((\{x_{11}, x_{12}\}, \{d_{11}, d_{12}\}), \{c_8\})$.

16.2.2.1 Loosely coupled Constraint Satisfaction Problems. In general case any CSP may be decomposed, either into a set of loosely coupled problems or into a set of strongly coupled problems (see Banaszak and Józefczyk (2005), Tomczuk et al (2005)). The two problems $CSP = ((X, D), C)$ and $CSP' = ((X', D'), C')$ are loosely coupled ones if the conditions 16.1 hold.

$$\begin{array}{ll} i) & X \cap X' = \emptyset \\ ii) & \forall c \in C : D(c) \cap X' = \emptyset \\ iii) & \forall c' \in C' : D(c') \cap X = \emptyset \end{array} \qquad (16.1)$$

where: $D(c)$ – is the set of variables included in the constraint c.

In turn, any element of a set of loosely coupled problems is a strongly coupled problem following the condition below

$$\begin{aligned}&\forall X_j^*, X_i^* \subset X^*, \exists X_a^*, X_b^*, \ldots, X_z^* \subset X^* : \\ &X_j^* \cap X_a^* \neq \emptyset \wedge X_a^* \cap X_b^* \neq \emptyset \wedge \ldots \wedge X_z^* \cap X_i^* \neq \emptyset\end{aligned} \tag{16.2}$$

where:

$CSP^* = ((X^*, D^*)C^*)$ – a strongly coupled problem composed a set of elementary problems $\{CSP_1^*, CSP_2^*, \ldots, CSP_k^*\}$.

It means, that for any two pairs of elementary problems of a strongly coupled problem there exists either nonempty subset of common variables or there exists a set of elementary subproblems constraints of which provide a chine of nonempty subsets of variables (following the pairs of elementary problems while linking the considered pair).

16.2.2.2 Dependent Constraint Satisfaction Problems. In turn, a strongly coupled problem may be decomposed into a set of so called dependent problems which are strongly coupled ones. It is assumed, however, that any pair of dependent problems follows the condition 16.3.

$$\begin{aligned}&\forall X_j^\wedge, X_i^\wedge \subset X, \exists X_k^* \subset X | X_j^\wedge \cap X_i^\wedge = \emptyset \wedge \ldots \wedge \\ &X_k^* \cap (\bigcup\{X'^* | X'^* \subset X_j^\wedge\}) \neq \emptyset \wedge X_k^* \cap (\bigcup\{X^* | X^* \in X_i^\wedge\}) \neq \emptyset\end{aligned} \tag{16.3}$$

where:

$CSP_j^\wedge = ((X_j^\wedge, D_j^\wedge), C_j^\wedge), CSP_i^\wedge = ((X_i^\wedge, D_i^\wedge), C_i^\wedge)$ - are the strongly coupled subproblems of the strongly coupled problem $CSP = ((X, D), C)$, $CSP^* = ((X*, D*), C*)$ - elementary subproblem of the problem $CSP = ((X, D), C)$.

Illustration of the $CSP = ((X, D), C)$ decomposition into the sets of loosely and strongly coupled as well as dependent subproblems is shown in Fig. 16.5.

16.2.2.3 Constraint Satisfaction Problem Decomposition . In order to summarize the above considerations it should be noted that a $CSP = ((X, D), C)$ can be decomposed into a set of:

- elementary subproblems,
- loosely coupled subproblems,
- dependent subproblems of a strongly coupled problems.

Instead of the first two ways of possible CSP decompositions the third one does not lead to a unique decomposition. For instance, besides of the possible decomposition shown in Fig. 16.6, an alternative the more detailed one can be considered as shown in Fig. 16.7.

Such observation enables to consider a tree of all the potentially available decompositions in a form of a AND/OR –like digraph, see Fig. 16.8.

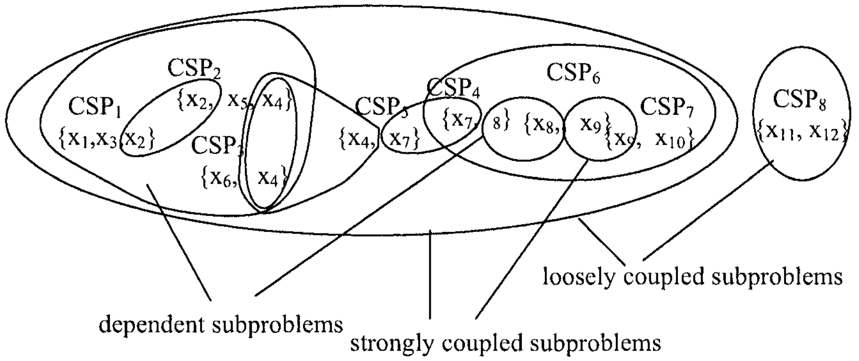


Figure 16.5. Decomposition of $CSP = ((X, D), C)$ into loosely and strongly coupled as well as dependent subproblems.

Different possibilities of CSP decomposition enable one to take into account the real life constraints that follow from:

- a way of a problem specification (i.e., a set of elementary problems recognized),
- a programming language implementation (some structures of dependent problems may or may not be accepted by CP/CLP packages),
- a way of a CSP resolution (e.g., the loosely coupled subproblems can be computed concurrently within an multiprocessor environment),
- a searching strategy applied (the order of subproblems resolution results in a CSP makespan).

The above observation leads to a concept of a reference model of a CSP decomposition, i.e., the model encompassing an object-like nature of the CSP structure (see Banaszak et al (2005), Tomczuk and Banaszak (2005)). So, since each subproblem corresponds to a standard constraint problem structure: (({a set of decision variables}, {a set of variable domains}), {a set of constraints}), hence some AND/OR – like graph representation can be used both in the course of analysis of the CSP programming (i.e. CP/CLP problem specification) and its resolution.

The concept of the CSP decomposition reference model provides a well suited framework for preliminary evaluation of search trees pruning strategies.

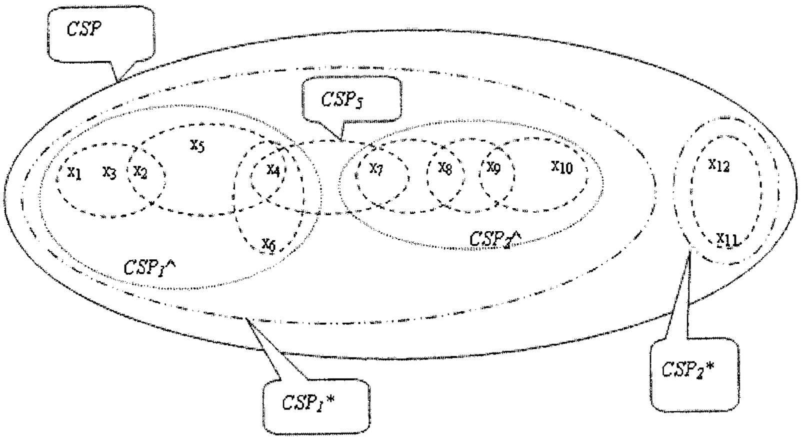


Figure 16.6. The possible decomposition of the CSP problem into the two loosely coupled subproblems CSP_1^*, CSP_2^*, and decomposition of the CSP_1^* into the two dependent subproblems $CSP_1^\wedge$ and $CSP_1^\wedge$.

16.3 State space pruning strategies

Consider the $CSP = ((X, D), C)$, where $X = \{x_1, x_2, x_3\}$, $D = \{D_1, D_2, D_3\}$, $C = \{c_1, c_2, c_3\}$, and where: $D_1 = D_2 = D_3 = \{1, 2\}$, $c_1 = P[x_1]$, $c_2 = P[x_2]$, $c_3 = P[x_3]$. The CSP consists of the following three elementary problems: $CSP_1 = ((\{x_1\}, \{D_1\}), \{c_1\})$, $CSP_2 = ((\{x_2\}, \{D_2\}), \{c_2\})$, $CSP_3 = ((\{x_3\}, \{D_3\}), \{c_3\})$.

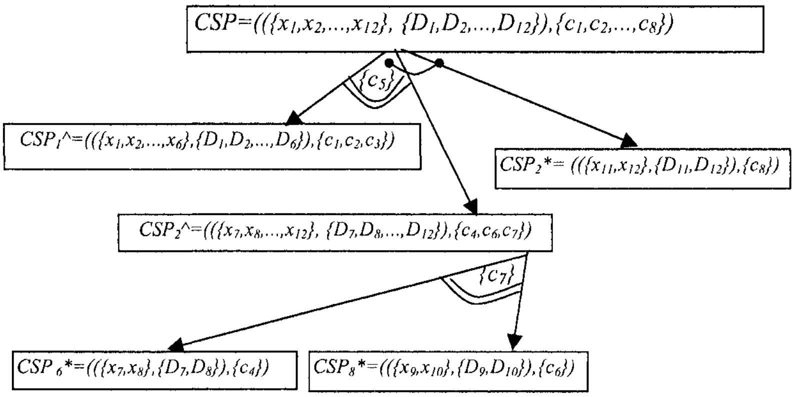
The number of possible solutions is equal to $2*2*2 = 8$, however the number of backtrackings required to check their feasibility is greater, and equals to 11 (see Fig. 16.8).

The number of backtrackings can be estimated due to the formulae (16.4).

$$N(j) = \sum_{i=1}^{L} \left(\prod_{h=1}^{i} Z_h^j - 1 \right) \tag{16.4}$$

where:

k – the index of the k-th elementary problem of a CSP,
L – the number of elementary problems,
j– the j-th permutation of the set of elementary problems,


CSP=(({x1,x2,...,x12}, {D1,D2,...,D12}),{c1,c2,...,c8})
{c5}
CSP1^=(({x1,x2,...,x6},{D1,D2,...,D6}),{c1,c2,c3})
CSP2*= (({x11,x12},{D11,D12}),{c8})
CSP2^=(({x7,x8,...,x12}, {D7,D8,...,D12}),{c4,c6,c7})
{c7}
CSP6*=(({x7,x8},{D7,D8}),{c4})
CSP8*=(({x9,x10},{D9,D10}),{c6})

Legend:
$CSP_2^*, CSP_6^*, CSP_8^*$ - elementary subproblems,
$CSP_1^\wedge, CSP_2\wedge, CSP_2^*, CSP_6^*, CSP_8^*$ - strongly coupled subproblems,
$CSP_1^* = ((\{x_1 - x_{10}\}, \{D_1 - D_{10}\}), \{c_1 - c_7\}), CSP_2^*$ - loosely coupled subproblems,

⌣ decomposition into dependent subproblems,
●⌣● decomposition into loosely coupled subproblems.

Figure 16.7. Alternative way of the CSP problem decomposition.

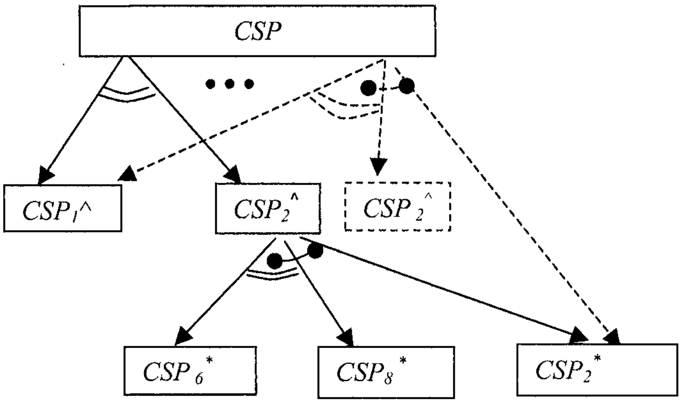


Figure 16.8. AND/OR-like graph representation of the CSP possible decompositions.

h – the index of an elementary problem placed at the h-th position in the j_k-th permutation of the set of elementary problem obtained from the CSP,

$Z_h^j(k)$– a number of potential assignments of the decision variables of the k-th elementary problem placed at the h–th position in the j_k-th permutation.

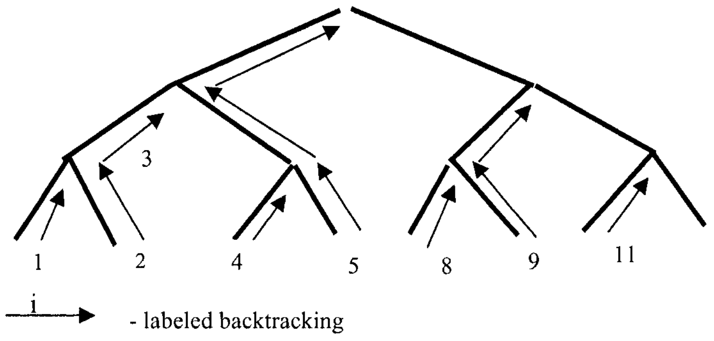


Figure 16.9. Searching tree encompassed by backtrackings.

So, assuming (CSP_2, CSP_1, CSP_3) as the j-th permutation of elementary problems $\{CSP_1, CSP_2, CSP_3\}$ as well as $Z_h^j(2) = Z_h^j(1) = Z_h^j(3) = 2$, the number of backtrackings $N(j)$ equals to: $1 + 2 * 2 - 1 + 2 * 2 * 2 - 1 = 11$.

16.3.1 Searching strategy evaluation

In general case, however, since the cardinality of a set of possible solutions of each elementary problem $CSP_1, CSP_2, \ldots, CSP_L$ of CSP can be seen as a multiple of its variables domains, hence the possible orders of CSP resolution are determined by $L!$ permutations of the set of elementary problems. Of course, the different permutations lead to the different results, i.e. different numbers of backtrackings.

In order to illustrate this fact let us consider the $CSP = ((X, D), C)$, where $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, $D = \{d_1, d_2, d_3, d_4, d_5, d_6\}$, $C = \{c_1, c_2, c_4\}$ and where: $D_1 = D_2 = D_3 = \{1, 2\}$, $D_4 = D_5 = D_6 = \{1, 2, 3\}$, $c_1 = P[x_1, x_2]$, $c_2 = P[x_3, x_4]$, $c_3 = P[x_5, x_6]$.

The CSP considered consists of the following three elementary problems: $CSP_1 = ((\{x_1, x_2\}, \{D_1, D_2\}), \{c_1\}), CSP_2 = ((\{x_3, x_4\}, \{D_3, D_4\}), \{c_2\})$, $CSP_3 = ((\{x_5, x_6\}), \{D_5, D_6\}), \{c_3\})$. Among the possible 3! permutations let us focus on the following two ones: (PSO_1, PSO_2, PSO_3), and (PSO_3, PSO_1, PSO_2).

Since CSP_1 results in possible $2*2 = 4$ solutions (assignments), and CSP_2 results in possible $2 * 3 = 6$ solutions, and CSP_3 results in possible $3 * 3 = 9$ solutions, hence first permutation result in $4 - 1 + 4 * 6 - 1 + 4 * 6 * 9 - 1 = 241$ backtrackings, and the second one in $9 - 1 + 9 * 4 - 1 + 9 * 4 * 6 - 1 = 258$ backtrackings.

In the considered case, however, the way of backtrackings estimation suffers from omitting the number of possible backtrackings at elementary problem levels. Note, that for the case when elementary problem consists of three and more variables a number of required backtracking is bigger than number of possible solutions to the problem.

In order to overcome this disadvantage the modified formulas are proposed (16.5), (16.6).

$$N(j) = \sum_{k=1}^{L} \left(\prod_{r=1}^{k} N_r^j(j_k, k) - 1 \right) \tag{16.5}$$

where:

L – the number of elementary problems of a CSP,

j – the j-th permutation of a set of the elementary problems of a CSP,

r – the index of an elementary problem placed at the r-th position in the j-th permutation,

k – the index of the k-th elementary problem,

j_k – the j_k-th permutation of a set of the k-th elementary problem variables,

$N_r^j(j_k, k)$ the number of potential backtrackings of the k-th elementary problem resolved due to the j_k-permutation of variables, the k-th elementary problem is placed at the r-th position in the j-th permutation.

$$N(j_k, k) = \sum_{i=1}^{L(k)} \left(\prod_{h=1}^{i} Z_h^{j_k}(i, k) - 1 \right. \tag{16.6}$$

where:

k – the index of the k-th elementary problem of a CSP,

$L(k)$ – the number of the variables of the k-th elementary problem,

i – the index of the i-th variable of the k-th elementary problem,

j_k– the k-th permutation of the variables of the k-th elementary problem,

h– the index of the variable placed at the h-th position in the j_k-th permutation of the k-th elementary problem variables,

$Z_h^{j_k}(i, k)$ the cardinality of the i-th variable domain, i.e., the variable placed at the h-the position in thej_k-th variables permutation of the k-th elementary problem

So, in order to obtain a correct evaluation of the backtrackings number required to find a set of admissible solutions of a CSP, the permutation of elementary problems as well as variables permutation in each elementary problem have to be assumed. The variables and elementary problems permutation determine the order of variables substitution and elementary problems resolution, respectively.

In the case considered, for the set of elementary problems $\{CSP_1$, CSP_2, $CSP_3\}$ let us consider:

- the permutation (CSP_1, CSP_2, CSP_3) and the following variables permutation: (x_1, x_2) for CSP_1, (x_3, x_4) for CSP_2, and (x_5, x_6) for CSP_3,

- the permutation (CSP_3, CSP_1, CSP_2) and the following variables permutation: (x_1, x_2) for CSP_1, (x_3, x_4) for CSP_2, and (x_5, x_6) for CSP_3.

In the first case, since CSP_1 results (see formulae (16.2)) in possible $1 + 2 * 2 - 1 = 4$ solutions (assignments), and CSP_2 results in possible $1 + 2 * 3 - 1 = 6$ solutions, and CSP_3 results in possible $2 + 3 * 3 - 1 = 10$ solutions, hence due to the formulae (16.3) the number of backtrackings equals to $4 - 1 + 4 * 6 - 1 + 4 * 6 * 10 - 1 = 275$ backtrackings. In the second case, however, the number of backtrackings equals to $10 - 1 + 10 * 4 - 1 + 10 * 4 * 6 - 1 = 287$ backtrackings.

Assuming a new variables permutation: (x_1, x_2) for CSP_1, (x_3, x_4) for CSP_2, and (x_5, x_6) for CSP_3, the relevant numbers of backtrackings equal to: 309, and 357, respectively.

The above observation providing a way of pruning strategies evaluation can be generalized for the case of loosely and strongly coupled subproblems of CSP. The formulas allowing one to evaluate the pruning strategies for a given CSP decomposition as well as for assumed subproblems and variables permutation are provided, see the formulae (16.7), and (16.8), respectively.

$$N(j_d) = \sum_{k=1}^{L} \left(\prod_{r=1}^{L} N_r^{j_d}(j_k, k) - 1 \right) \tag{16.7}$$

where:

L – the number of the subproblems obtained due to the d-th decomposition of a CSP,

j_d – the j_d-th permutation of a set of subproblems obtained due to the d-th decomposition of a CSP,

r – the index of the subproblem placed in the r-th position in the j_d-th permutation,

k – the k-th subproblem of a set of subproblems obtained due to the $d-$th decomposition of a CSP,

j_k – the j_k-th permutation of a set of the k-th subproblem variables,

$N_r^{j_d}(j_k, k)$ the number of potential backtrackings of the k-th subproblem placed at the r-th position in the j_d-th permutation

$$N(j_k, k) = \sum_{i=1}^{L(k)} \left(\prod_{h=1}^{L(k)} Z_h^{j_k}(i, k) - 1 \right) \tag{16.8}$$

where:

k – the k-th subproblem of a set of subproblems obtained due to the d-th decomposition of a CSP,

$L(k)$ – the number of the variables of the k-th subproblem,

i – the index of the i-th variable of the k–th subproblem,

j_k– the k-th permutation of the variables of the k-th subproblem,

h– the index of the variable placed at the$h$-th position in the j_k-th permutation of the k-th subproblem variables,

$Z_h^{j_k}(i,k)$ the cardinality of the i-th variable domain, i.e., the variable placed at the h–th position in the j_k-th variables permutation of the k-th subproblem

16.3.2 Searching for searching strategy

In order to summarize the section it should be noticed that since with arcs of a AND/OR graph it is possible to bind weight factors determining the necessary number of searches, hence such representation provides a way to chose the best searching strategy, i.e. a variant with least number of backtrackings. In the case of a CSP decomposition into a set $\{CSP_1, CSP_2, \ldots, CSP_L\}$ the relevant searching tree is shown in Fig. 16.10.

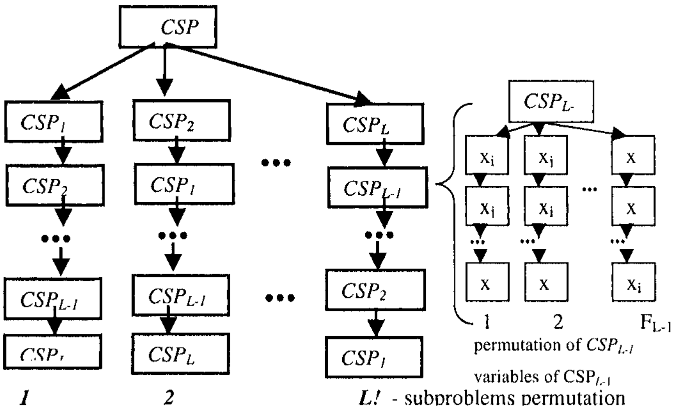


Figure 16.10. Searching tree for CSP decomposed into the set of elementary problems.

In the case considered the number V of possible searching strategies for the CSP composed of L elementary problems consisting of the $K_1, K_2, \ldots, K_L$ variables can be estimated due to the upper bound stated by the formulae (16.9). It means that for the CSP consisting of $L = 10$ elementary problems each of them containing $K_i = 2$ variables the number of possible searching is equal to $V = 2^{10}10! \approx 3.6 * 10^7$. Of course the formula considered does not take into account a number of possible CSP decompositions!

$$V = (\max\{K_i\})^L L! \tag{16.9}$$

Therefore the problem of selection of the optimal (i.e. requiring the less backtrackings) state space pruning strategy seems to be at least NP-complete one. So, the branch and bound method can be considered in the course of the best searching strategy selection. The idea standing behind of this concept is shown in Fig. 16.11.

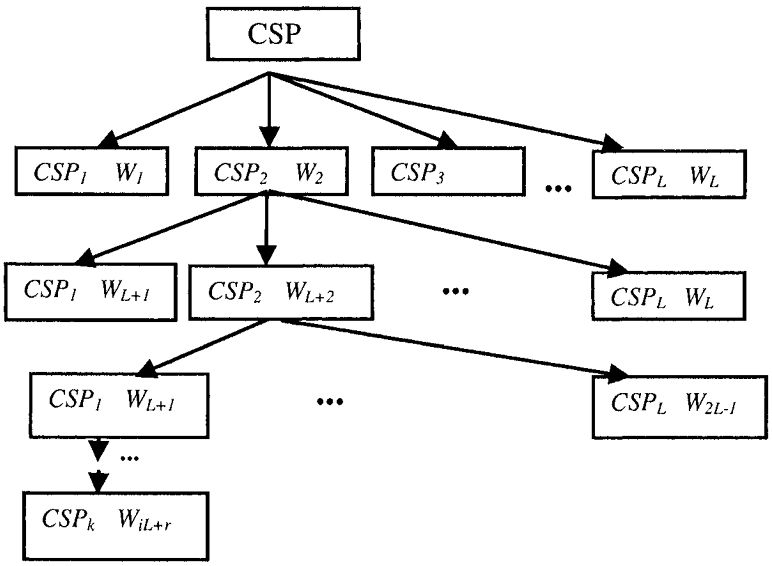


Figure 16.11. Branch and bound method approach to pruning strategy selection.

For a given CSP decomposition, i.e., for a given set of subproblems $\{CSP_1, CSP_2, \ldots, CSP_L\}$ the set of upper bound values $\{W_1, W_2, \ldots, W_L\}$ is calculated (due to the formulae (16.7) and/or (16.8). Then for the subproblem to which the lowest value of the upper bound is assigned the next subproblem is selected as to find the order in which the subproblems should be resolved while requiring the lowest number of backtrackings.

In general case, instead of the upper bound considered till now the other measures (heuristics) could be taken into account. For the illustration of such possibility let us consider the following example.

Given a $CSP = ((X, D), C)$ such that $X = \{x_1, x_2, x_3, x_4, x_5\}$, $D = \{D_1, D_2, D_3, D_4, D_5\}$, $D_1 = D_2 = D_3 = D_4 = D_5 = \{1, 2, 3, \ldots, 100\}$, $C = \{c_1, c_2, c_3, c_4\}$, $c_1 = P[2 * x_1 + x_2 \leq x_3]$, $c_2 = P[2 * x_2 = x_4], c_3 = P[x_4 * x_5 \leq x_1 * x_3]$, $c_4 = P[x_3 * x_4 * x_5 \leq 300]$.

As result of constraints propagation, i.e., the reduction of the domains $D_1 = \{1, 2, \ldots, 36\}, D_2 = \{1, 2, \ldots, 24\}, D_3 = \{3, 4, \ldots, 74\}, D_4 = \{4, 5, \ldots, 96\}$, $D_5 = \{1, 2, \ldots, 24\}$ the state space size of 100^5 is reduced to the size equal to 138 848 256. The constraint influence on the state space size reduction is shown in the Table 16.1.

Table 16.1. The state space size reduction influenced by constraints.

Con-	Domains of decision variables					The rate of the state space
straints	D_1	D_2	D_3	D_4	D_5	reduction
c1	1 - 49	1 - 98	3 - 100	1 - 100	1 - 100	52,94 %
c2	1 - 100	1 - 25	1 - 100	4 - 100	1 - 100	75,75 %
c3	1 - 100	1 - 100	1 - 100	1 - 100	1 - 100	0 %
c4	1 - 100	1 - 100	1 - 100	1 - 100	1 - 100	0 %

Using the results obtained one may consider a searching strategy employing the order of constraints propagation. Such strategy assumes step by step elementary problems resolution emphasizing a dynamic of state space reduction (i.e., a heuristics assuming: "faster state space reduction, shorter searching time"). The evaluation of the possible strategies is shown in the Table 16.2.

The elements of the third, fifth, seventh and the last column in the Table 16.2 are calculated as follows:

- the third column $[1 - (49 \cdot 98 \cdot 98 \cdot 100 \cdot 100)/(100^5)] \cdot 100\% = 52,94\%$ for the constraint c_1
- the fifth column $[1 - (49 \cdot 25 \cdot 98 \cdot 97 \cdot 100)/(100^5)] \cdot 100\% = 88,35\%$ for the constraint c_3
- the seventh column $[1 - (49 \cdot 25 \cdot 98 \cdot 97 \cdot 100)/(100^5)] \cdot 100\% = 88,35\%$ for the constraint c_3
- the last column $[1 - (49 \cdot 25 \cdot 98 \cdot 97 \cdot 100)/(100^5)] \cdot 100\% = 88,35\%$ for the constraint c_4

Note that nonlinear constraints are less effective in the state space size reduction than the linear ones.

16.4 Towards task oriented decision support tool designing

Since the reference model, i.e. an object-like AND/OR graph framework, provides the possibility to estimate the number of decision variables domains values substitution, hence the influence of data structure, sequence of elementary subproblems solution and domain size on decision making time may be

Table 16.2. The state space reduction pruning strategies based on the step by step constraints propagation.

	Constraints and corresponding state space reduction rate							
Searching	Searching strategy							
strategy		%		%		%		%
1	c1	52,49	c2	88,35	c3	88,35	c4	98,61
2	c1	52,49	c2	88,35	c4	98,61	c3	98,61
3	c1	52,49	c3	52,94	c2	88,35	c4	98,61
4	c1	52,49	c3	52,94	c4	53,87	c2	98,61
5	c1	52,49	c4	53,87	c2	98,61	c3	98,61
6	c1	52,49	c4	53,87	c3	53,87	c2	98,61
7	c2	75,75	c1	88,35	c)	88,35	c4	98,61
8	c2	75,75	c1	88,35	c4	98,61	c3	98,61
9	c2	75,75	c3	75,75	c1	88,35	cd	98,61
10	c2	75,75	c3	75,75	c4	86,72	c1	98,61
11	c2	75,75	c4	86,72	c1	98,61	c3	98,61
12	c2	75,75	c4	86,72	c3	86,72	c1	98,61

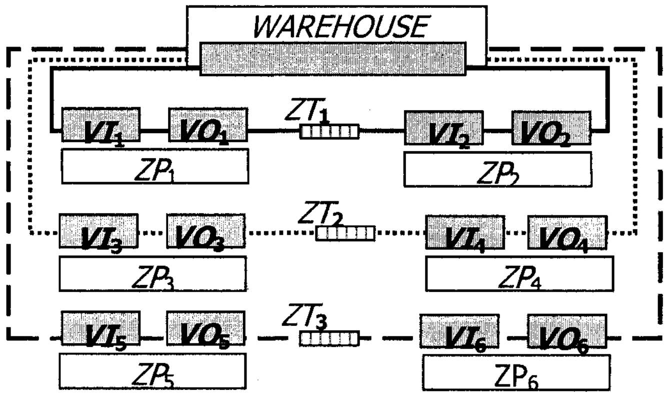
evaluated as well. In other words, the model considered provides a well suited framework for development (taking into account the ways of possible problem specification, available CP/CLP languages, and searching strategies) of a CP-based programming methodology as well as the development of the task oriented software tools aimed at the SMEs decision support, e.g. regarding production flow planning.

16.4.1 Problem Formulation

Many often repeating question regards of the question: Whether in a given shop employed with the machine tools, automated guided vehicles (AGVs), and buffers and warehouses a production order submitted can be completed due assumed period? A production order includes the type of final product to be manufactured, the required quantity and defined due date as well as a final cost a customer has to pay. The product type is usually defined as the combination of components that a machine toll is capable of handling. The product type is specified by an execution route (production routing) that includes one or more execution steps. In general case, however, a given final product can be produced differently due to alternative execution routes that differ in the execution steps.

The question stated above is usually considered under assumption in the enterprise executes several project-like production orders, however some production capability are still available. So, the question is whether the production order at hand can be accepted for execution in an enterprise where some resources availability is constrained in time?

The typical workshop layout (see Fig. 16.12) enables to distinguish two kinds of flows regarding production and transportation routings, respectively.



Legend:

ZP_j – the i-th machine tool, VI_j/VO_j– the i-th input/output buffer, ZT_i – *the i-th* AGV.

Figure 16.12. The workshop layout

These flows interact each other (e.g., via production and batch sizes, AGVs, buffers and warehouses capacities, and so on) and falls into the following two main subproblems: manufacturing and transportation that may be resolved in two alternative orders (see Fig. 16.13).

The possible decomposition of the production flow planning is shown in Fig 16.14. So, the relevant CSP problem consists of two subproblems corresponding to the manufacturing (denoted by CSP_1) and transportation (denoted by **G**) subflows. In turn, the subproblem of manufacturing falls into production routing (denoted by **A**) and production batch planning and scheduling (denoted by CSP_2). The last subproblem falls into subproblem of scheduling (denoted by **F**), and production batch planning (denoted by $CSP_{2,1}$). Finally, the production batch planning subproblems falls into the calculation of the number of production batches (denoted by **B**) and production batch sizing (denoted by **E**).

There are growing needs for decision support tools capable to assist a decision maker in the course of a new production order evaluation. The possible approach based on CP/CLP paradigm assumes possibility to adjust the ways of problem specification, its programming language implementation as well as a searching strategy selection to a class of production planning problems (specified by scale, production type, e.g. assembly or machining, and so on) as to respond in on line mode.

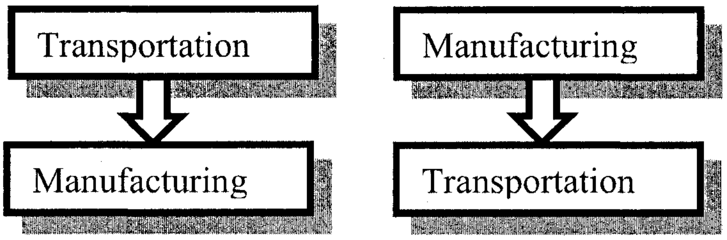


Figure 16.13. The possible orders of production flow subproblems resolution.

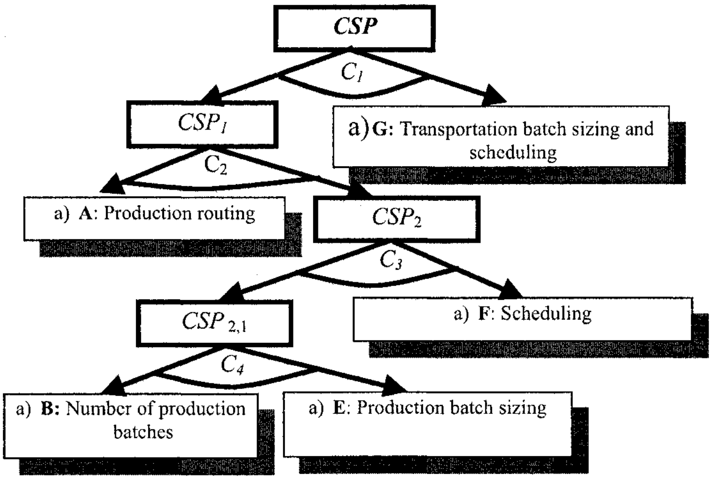

CSP
C1
CSP1
a) G: Transportation batch sizing and scheduling
C2
a) A: Production routing
CSP2
C3
CSP2,1
a) F: Scheduling
C4
a) B: Number of production batches
a) E: Production batch sizing

Figure 16.14. Decomposition of the production flow planning problem.

16.4.2 Task Oriented Decision Support Tool Designing

Programming methodology proposed takes into account the constraints imposed by a programmer experience (possible problem statements), by a set of available software tools (CP/CLP languages), and a set of searching strategies (build-in the software tools as well as those proposed by programmers) (see Fig.16.15). The idea standing behind of this approach assumes that a decision maker has to be supported in the course of a standard requests and regarding known in advance a class of situations that may occur in the SME at hand.

Following these requirements a programmer has to develop a well adjusted CP/CLP based decision support tool encompassing a specific of an enterprise and production orders considered.

The illustration of the implementation of the methodology considered into the task oriented software tools supporting the SME in the course of decision making is shown in Fig. 16.16. The way of admissible problem resolution is underlined by the bold frames and arcs.

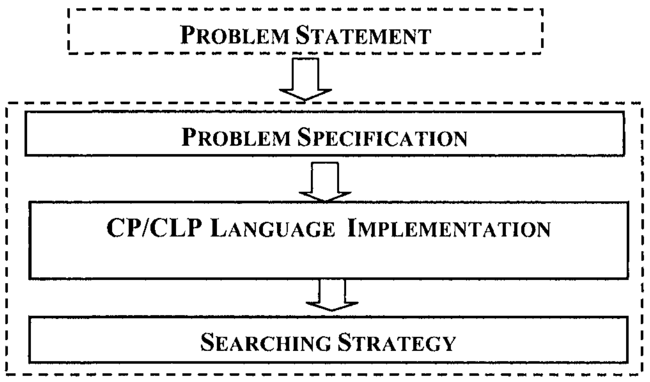


Figure 16.15. Stages of the CP-based programming methodology.

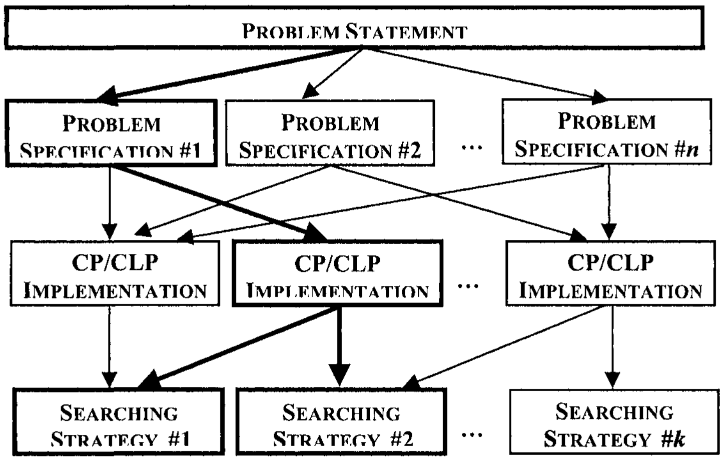


Figure 16.16. The tree of possible ways of a CSP programming.

Since efficiency of programming depends on selected problem specification, a software tool employed and a chosen searching strategy, hence designing of the task oriented tools seems to be the only reasonable approach to set-up a CP-based decision support software. It means, for a given class of CSPs (e.g.

production flow planning ones) and a CP/CLP language assumed, on the base of an experience gathered both from the analysis of the reference model of CSP decomposition, and multiple experiments that is possible to develop searching strategy optimal in the sense of minimal number of potential backtrackings.

16.4.3 Illustrative Example

The methodology presented has been implemented in the course of the development of a task oriented decision support tool aimed at tasks relevant to project management. The software package considered (implemented in ILOG OPL Studio 3.7) assists the user in answering to the following questions in the on-line mode: Are the company production capabilities are sufficient for the execution of a production order in accordance with the customer's requirements? What is the planned production order execution deadline? What is the cost of the order execution?

Moreover it facilitates setting a possible variant of current production organization including routing, batching and scheduling. Also it enables a fast evaluation of production orders assuming to be processed in a system already involved in the currently executed production plan. So, a newly introduced production orders are verified according to the company's capabilities and producer's requirements, e.g. the transportation-warehouse capability and the customer's requirements (the directive deadline, the cost and the production volume).

In order to illustrate its application (at a SME manufacturing the hydraulic and pneumatic equipment (see Tomczuk and Banaszak (2004), Tomczuk and Banaszak (2005)) let us consider the following example regarding of three production orders B_1, B_2, and B_3 see the Table 16.3.

Table 16.3. Specification of production orders

Production order	Work piece	Number of operations involved	Production volume(pcs.)	Suggested price (cost units)	Completion time (time units)
B1	Filter set	10	100	2100	2600
B2	Main body	27	25	4600	5500
B3	Valve	7	20	1700	1800

A specification of each production order covers subsequent operations and their execution times as well as resources required. For instance in the case of the production order B_1 regarding of the filter set (consisting of a filter and a connector) manufacturing the relevant data are specified see the Table 16.4.

It is assumed, after completion of each production operation a transportation operation to the next workstation along a given technological production route

Table 16.4. Specification of the operations and their operation times in production order B_1

Operation name	Number of the subsequent operation	Execution time (time unit/pcs.)	tpz_{A_j}	Resource
Operations in the execution of the filter				
Cutting	A_1	1	50	R_8
Washing	A_2	1	120	R_2
Control	A_3	3	20	R_{19}
Operations in the execution of the connector				
Cutting	A_4	1	20	R_{12}
Turning	A_5	1	90	R_{13}
Washing	A_6	1	60	R_2
Turning	A_7	2	110	R_{27}
Hand treatment	A8	1	50	R_7
Washing	A_9	1	60	R_{31}
Blacking	A_{10}	2	60	R_{17}

Legend:
A_j the j-th operation, $j = 1, \ldots, 10$,
R_i– the i-th resource (work place), $i = 1, \ldots, 32$,
tpz_{A_j} the preparation-finishing time.

is executed. The transportation means, their capacity, transportation routings and the duration times are defined as well.

For the data included in the Table 16.4 the following sequence of execution of production orders B_2, B_1, B_3 is considered at first. The sequence in which the production orders follow each other provides the priority enabling conflicts resolution. Due to the system's capability following from the currently realized production plan, all the production orders considered cannot be accepted. That is because the production order B_1 cannot be processed see the Table 16.5.

The next sequence B_1, B_2, B_3, however, facilitates acceptance of all orders for their execution. The plan obtained provides the time of starting the production order B_1 at 1 time unit, the production order B_2 at 151 time unit, and B_3 at 1 time unit see the Table 16.5.

The solution obtained takes into account the possibility of execution of a production order due to the technological sequence of operations, transportation routings among resources and the production volume, capacity of buffers and

Table 16.5. Results of computer based experiments

Variants	Production order	Solution	Beginning moment [time unit]	Completion time [time unit]	Total cost [cost unit]	Computation time [sec.]
	B_1	No solution	-	exceeded	-	
B_2, B_1, B_1	B_2	Obtained	1	5101	4489.05	1.2
	B_3	Obtained	1	1661	1444.90	
	B_1	Obtained	1	2279	1875.70	
B_1, B_2, B_3	B_2	Obtained	151	5483	4489.05	1.4
	B_3	Obtained	1	1762	1444.90	

Legend:

B_2, B_1, B_1– work orders: B_2 and B_3 may be executed in the system, work order B_1 cannot be executed (exceeded directive execution time).

B_1, B_2, B_3– work orders: B_1, B_2 and B_3 may be executed in the system.

their allocation. Moreover, knowing the resources availability the cost of the production order execution can be easily estimated as well.

For a given class of SMEs the software package considered enables one to deal with temporal constraints, priority between constraints and constraints due to the structure of the transportation paths in an on-line mode. Therefore, the example provided illustrates the versatility and adequacy of the CLP approach to the task oriented decision support tools designing, especially in the area of project management.

16.5 Concluding remarks

A CP/CLP – based modeling framework provides a good platform for consistency checking between the production order completion requirements and a workshop capability offered. Particularly it can be used for development of a project management decision support tools. The first prototype has been tested on the SME with very good results. In that context the CP/CLP methodology presented here seems to be a promising alternative for commercially available tools based on other technologies, such as a class of ERP systems. Their application in solving a real-life problem is quite limited (see Banaszak and Pisz (2003)).

The discussion provided has shown the versatility of CP/CLP paradigm for the scheduling problems. In particular, CP/CLP on finite discrete domains is a flexible and declarative paradigm for solving many scheduling problems such as project-like manufacturing (e.g. a unique and/or short batch production), subject to many temporal, priority and resource constraints. The consistency techniques and the forward propagation of constraints greatly reduce the search space of the problem and therefore make CP/CLP an effective tool for facing the project-driven manufacturing.

Therefore, the proposed approach can be considered as a contribution to project-driven production flow management applied in make-to-order companies as well as for prototyping of the virtual enterprises. That is especially important in the context of a cheap and user-friendly decision support for the SMEs. Further research is aimed at the development of the task oriented searching strategies, implementation of which could interface a decision maker with a user-friendly intelligent support system.

References

Anavi-Isakow, S. and Golany, B. (2003). Managing multi-project environments through constant work-in-process, *International Journal of Project Management* 21:9–18.

Banaszak, Z., Zaremba, M., Muszyński, W. (2005). CP-based decision making for SME, in: *Preprints of the 16th IFAC World Congres*, P. Horacek, M. Simandl and P. Zitek, eds, 3 – 8 July, 2005, Prague, Czech Republic, DVD.

Banaszak, Z., and Zaremba, M. (2004a). CLP-based project-driven manufacturing, *Prep. of the 7th IFAC Symposium on Cost Oriented Automation*, June 6-9, 2004, Gatineau, Quebec, Canada, pp. 269–274.

Banaszak, Z., and Zaremba, M. (2004b). Project-driven management support for web-enabled manufacturing, in: *Prep. of the 11th IFAC Symposium on Information Control Problems in Manufacturing*, C.E. Pereira, ed, Salvador, Brasil, 5-7 April, 2004, CD-ROM.

Banaszak, Z., Skołud, B. and Zaremba, M. (2003). Computer-aided prototyping of production flows for a virtual enterprise. *Journal of Intelligent Manufacturing* 14:83–106.

Banaszak, Z. (2003). Workflows management for project-driven manufacturing. *Prep. the 7th IFAC Workshop on Intelligent Manufacturing Systems*, 6-8 April, 2003, Budapest, Hungary, pp. 149–154.

Banaszak, Z. and Pisz, I. (2003). Project-driven production flow management, in: *Project-Driven Manufacturing*, Z. Banaszak and J. Józefowska, eds, WNT, Warsaw, pp. 53–71.

Banaszak, Z. and Józefczyk, J. (2005). Towards CLP-based task oriented DSS for SME, *Applied Computer Science and Production Management* 1:161–180.

Barták, R. (2004). Incomplete depth-first search techniques: a short survey, in: *Proceedings of the 6th Workshop on Constraint Programming for Decision and Control*, J. Figwer, ed, June 29, 2004, Gliwice, Poland, pp. 7–14.

Barták, R. (2003). Constraint-based scheduling: an introduction for newcomers, *Prep. of the 7th IFAC Workshop on Intelligent Manufacturing Systems,* 6-8 April, 2003, Budapest, Hungary, pp. 75–80.

Barták, R. (1998). On-line guide to constraint programming, Prague, http://kti.mff.cuni.cz/ ~ bartak/constraints/.

Blazewicz, J., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1983). Scheduling subject to resource constraints, *Discrete Applied Mathematics* 5:11–24.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: notation, classification, models and methods, *European Journal of Operational Research* 112:3–41.

Chin-Yin, H. (2002). Distributed manufacturing execution systems: A workflow perspective, *Journal of Intelligent Manufacturing* 13:485-497.

Kis, T., Ërdos, G., Màrkus, A. and Vàncza, J. (2004). *A Project-Oriented Decision Support Systems for Production Planning in Make-to-Order Manufacturing*, ERCIN News, Bo.58.

Lamma, E., Mello, P. and Milano, M. (1997). A distributed constraint-based scheduler, *Artificial Intelligence* 11:91–105.

Leach, L.P. (2000). *Critical chain project management*, Artech House, London.

Lova A., Maroto, C. and Tormos, P. (2000). A multicriteria heuristic method to improve resource allocation in multiproject scheduling, *European Journal of Operational Research* 127:408–424.

Payne, J.H. (1995). Management of multiple simultaneous projects: State-of-art review, *International Journal of Project Management* 13:163–168.

Ratchev, S. M., Shiau, J. and Valtchanov, G. (2000). Distributed product and facility prototyping in extended manufacturing enterprises, *International Journal of Production Research* 38:4495–4506.

Rossi, F. (2000). Constraint (Logic) programming: A Survey on Research and Applications, in: *New Trends in Constraints*, K.R. Apt et al, eds, LNAI 1865, Springer-Verlag, Berlin, pp. 40–74.

Tomczuk, I. and Banaszak, Z. (2005). Production flow planning based on CLP approach, *Computer Integrated Management*, R. Knosala, ed, WNT, Warszawa, pp. 589–600. (in Polish)

Tomczuk, I. and Banaszak, Z. (2004). Constraint programming approach for production flow planning, *Proceedings of the 6th Workshop on Constraint Programming for Decision and Control*, June 29, 2004, Gliwice, Poland, pp. 47–54.

Tomczuk, I., Bzdyra, K. and Banaszak, Z. (2005). Towards CLP-based task-oriented DSS for SME, *Applied Computer Science and Production Management* 1:181–200.

Van Henteryck, P., Perron, L. and Puge, J. (2000). Search and Strategies in OPL, *ACM Transactions on Computational Logic* 1:1–36.

Wallace, M. (2000). Constraint Logic Programming, In Kakas A.C., and Sadri F., editors, *Computational Logic*, LNAI 2407, Springer-Verlag, Berlin, Heidelberg, pp. 512–532.

Wei, C. C., Liu, P.-H. and Tsai, Y.-C. (2002). Resource-constrained project management using enhanced theory of constraint, *International Journal of Project Management* 20:561–567.

Index