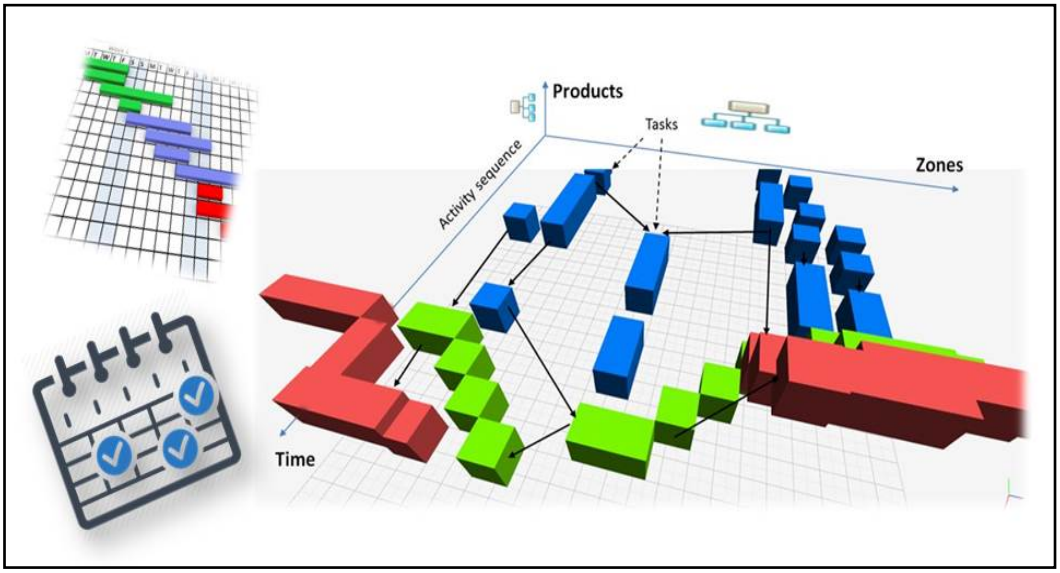
Scheduling problems with production and consumption of resources

Abderrahim Sahli

Par **Abderrahim SAHLI**

Problèmes d'ordonnancement avec production et consommation des ressources

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 20 octobre 2016
Spécialité : Technologies de l'Information et des Systèmes :
Unité de recherche Heudyasic (UMR-7253)
D2309

Problèmes d'ordonnancement avec production et consommation des ressources

Abderrahim SAHLI

Thèse soutenue le 20 Octobre 2016 devant le jury composé de :

Président:

Emmanuel NERON
Professeur des universités
Université François Rabelais de Tours

Rapporteurs:

Eric PINSON
Professeur des universités
Université Catholique de l'Ouest

Alain QUILLIOT
Professeur des universités
Université Blaise Pascal

Examinateurs:

Philippe LABORIE
Docteur
France Lab, IBM

Antoine JOUGLET
Maître de Conférences
Université de Technologie de Compiègne

Directeurs de Thèse:

Jacques CARLIER
Professeur des universités
Univ de Technologie de Compiègne

Aziz MOUKRIM
Professeur des universités
Univ de Technologie de Compiègne

Technologies de l'Information et des Systèmes

Sorbonne universités, Université de Technologie de Compiègne,
CNRS, laboratoire Heudiasyc UMR 7253, CS 60 319,
57 avenue de Landshut 60 203 Compiègne

20 - 10 - 2016

Scheduling problems with production and consumption of resources

Abderrahim SAHLI

20-10-2016

Avant propos

Cette thèse a été réalisée dans le laboratoire Heudiasyc, au sein de l'équipe RO. Son financement a été assuré par le laboratoire d'excellence Labex MS2T.

Je tiens à exprimer mes plus sincères remerciements à mes directeurs de thèse, Pr. Jacques CARLIER et Pr. Aziz MOUKRIM pour la confiance qu'ils m'ont accordée pour réaliser ce travail, pour leurs conseils et leur soutient. Je leur suis très reconnaissant de m'avoir permis de poursuivre mes recherches tout en étant toujours présent pour m'aider.

Je remercie M. Emmanuel NERON, Professeur à l'Université François Rabelais de Tours, pour avoir présidé mon jury de thèse. Mes remerciements vont également à Eric PINSON Professeur à l'Université Catholique de l'Ouest et Alain QUILLIOT Professeur à l'Université Université Blaise Pascal pour le temps passé à relire cette thèse en tant que rapporteurs et pour leurs remarques pertinentes vis à vis de mon travail. Je remercie également Antoine JOUGLET Maitre de conférence à l'Université de Technologie de Compiègne et Philippe LABORIE ingénieur de recherche chez IBM qui m'ont fait l'honneur d'accepter de participer au jury de ma thèse.

Je voudrais également remercier plus particulièrement mes parents et mes soeurs pour leur soutient et leur confiance en moi. Le parcours que j'ai fait jusqu'à aujourd'hui n'aurait pas été possible sans eux.

Enfin, je remercie mes collègues du laboratoire Heudiasyc pour la convivialité ainsi que la bonne ambiance qui régnait au laboratoire. Je remercie particulièrement mon ami Kaci BADER pour son aide et ses conseils.

Abstract

Title : Scheduling Problems with Production and Consumption of Resources.

This thesis investigates the Extended Resource Constrained Project Scheduling Problem (ERCPSP). ERCPSP is a general scheduling problem where the availability of a resource is depleted and replenished at the occurrence times of a set of events. It is an extension of the Resource Constrained Project Scheduling Problem (RCPSP) where activities are replaced by events, which have to be scheduled subject to generalized precedence relations.

We are interested in this thesis in proposing new methodologies and approaches to solve ERCPSP. First, we study some polynomial cases of this problem and we propose a dynamic programing algorithm to solve the parallel chain case. Then, we propose lower bounds, mixed integer programming models, and a branch-and-bound method to solve ERCPSP. Finally, we develop an instance generator dedicated to this problem.

Key Words : *Scheduling problem, nonrenewable resource, lower bounds, branch-and-bound, linear programming, dynamic programming.*

Supervisors : Jacques CARLIER and Aziz MOUKRIM.

Resumé

Titre : Problèmes d'ordonnancement avec production et consommation des ressources

La plupart des travaux de recherches sur les problèmes d'ordonnancement traitent le cas des ressources renouvelables, c'est-à-dire des ressources qui sont exigées en début d'exécution de chaque tâche et sont restituées en fin d'exécution. Peu d'entre eux abordent les problèmes à ressources consommables, c'est-à-dire des ressources non restituées en fin d'exécution. Le problème de gestion de projet à contraintes de ressources (RCPSP) est le problème à ressources renouvelables le plus traité dans la littérature.

Dans le cadre de cette thèse, nous nous sommes intéressés à une généralisation du problème RCPSP qui correspond au cas où les tâches sont remplacées par des événements liés par des relations de précédence étendues. Chaque événement peut produire ou consommer une quantité de ressources à sa date d'occurrence et la fonction économique reste la durée totale à minimiser. Nous avons nommé cette généralisation ERCPSP (Extended RCPSP). Nous avons élaboré des modèles de Programmation Linéaire pour résoudre ce problème. Nous avons proposé plusieurs bornes inférieures algorithmiques exploitant les travaux de la littérature sur les problèmes cumulatifs. Ensuite, nous avons élargi la porté des méthodes utilisées pour la mise en place de méthodes de séparation et évaluation. Nous avons traité aussi des cas particuliers par des méthodes basées sur la programmation dynamique.

Mots clés : *problème d'ordonnancement, ressource consommable, bornes inférieures, méthode arborescente, programmation linéaire, programmation dynamique.*

Directeurs de thèse : Jacques CARLIER et Aziz MOUKRIM.

Publications

Article journal

[1] J. Carlier, A. Moukrim, **A. Sahli**, Lower bounds for the Event Scheduling Problem with Consumption and Production of Resources, Discrete Applied Mathematics, 2016.

Conférences internationales avec actes

[2] **A. Sahli**, J. Carlier, and A. Moukrim, Lower bounds for Scheduling Problem with production and Consumption of Resources. In MISTA 2015, Prague, Czech Republic, August 2015.

[3] **A. Sahli**, J. Carlier, and A. Moukrim, A new LP-Based lower bound for the Event Scheduling Problem with Consumption and Production of Resources, In PMS 2016, Valence, Spain, April 2016.

[4] **A. Sahli**, J. Carlier, and A. Moukrim, Comparison of mixed integer linear programming models for the Event Scheduling Problem with Consumption and Production of Resources, In MIM 2016, Troyes, France, June 2016.

Autres communications

[5] J. Carlier, A. Moukrim, and **A. Sahli**, Nouvelles bornes pour un problème d ?ordonnancement avec production et consommation de ressources. In JOPT2014, Journées de l ?Optimisation, Montreal, Canada, May 2014.

[6] J. Carlier, A. Moukrim, and **A. Sahli**, Scheduling problems with production and consumption of resources. In Ninth International Colloquium on Graphs and Optimization, Sirmione, Italy, July 2014.

[7] J. Carlier, A. Moukrim, **A. Sahli**. Du générique au spécifique : Problèmes d ?ordonnancement avec production et consommation des ressources. In ROADEF 2016, Compiegne, France, Feb 2016.

En cours de rédaction

[8] **A. Sahli**, J. Carlier and A. Moukrim. Polynomial cases of the Extended RCPSP.

[9] **A. Sahli**, J. Carlier and A. Moukrim. A Branch-and-bound method for the Extended RCPSP.

Table des matières

Avant propos i

Abstract iii

Resumé v

Publications vii

Table des matières ix

Liste des figures xiii

Liste des tableaux xv

Liste des algorithmes xvii

Notations xix

Introduction 1

1 Scheduling Problems 5

1.1 Introduction 5

1.2 Definition of Scheduling Problems 6

1.2.1 Activities and Events 6

1.2.2 Resources 7

1.2.3 Precedence relations 7

1.2.4 Constraints and Objectives 8

1.3 Resource Constrained Project Scheduling Problem 8

1.3.1 Problem Description 8

1.3.2 Lower bound calculations 9

1.3.2.1 Critical Path and Capacity Bounds 9

1.3.2.2 Bin Packing Bounds 9

1.3.2.3 Node Packing Bounds 10

1.3.2.4 Parallel Machine Bounds 10

1.3.2.5 Precedence Bounds 10
1.3.2.6 LP-Based Bounds 10
1.3.3 Exact procedures 11
1.3.3.1 Linear programming based approaches 11
1.3.3.2 Branch-and-bound procedures 11
1.3.3.3 Chronological branching scheme 13
1.3.3.4 Minimal forbidden sets 14
1.3.4 Heuristic procedures 14
1.4 Scheduling problems with non-renewable resources 15
1.4.1 The Financing Problem 15
1.4.1.1 Problem description 15
1.4.1.2 Feasible Schedule 15
1.4.1.3 The Shifting Algorithm 16
1.4.2 The Project Scheduling Problem with Inventory Constraints . 16
1.4.2.1 Problem description 16
1.4.2.2 The Exact Method of Neumann and Schwindt 17
1.4.2.3 The Exact Method of Laborie 18
1.4.3 RCPSP with Consumption and Production of Resources . . . 19
1.5 Conclusion 20

2 The Extended RCPSP 23
2.1 Introduction 23
2.2 Problem formulation 25
2.2.1 Project Scheduling Problem with Inventory Constraints and ERCPSP 27
2.2.2 RCPSP/CPR and ERCPSP 28
2.3 Decision and Resource Usage Problems 29
2.4 Linear orders for ERCPSP 29
2.4.1 Compatibility 30
2.4.2 EST schedule of a complete linear order 31
2.4.3 Case of multiple resources 31
2.4.4 Linear order of consumption events 32
2.5 Polynomial cases of ERCPSP 32
2.5.1 The Relocation Problem 32
2.5.1.1 Feasibility test of Kaplan and Amir 33
2.5.1.2 Johnson's rule 34
2.5.2 The parallel chain case 35
2.5.2.1 Notation 36

2.5.2.2 Decomposition of chains into optimal subchains . . . 37
2.5.2.3 Standard Form Schedule . . . 40
2.5.2.4 Algorithm of resolution . . . 43
2.5.3 The series-parallel case . . . 45
2.5.4 The interval order case . . . 47
2.5.4.1 Interval orders . . . 47
2.5.4.2 List schedule for interval order case . . . 49
2.6 Dynamic Programming: parallel chain case . . . 50
2.6.1 Dynamic programming approach . . . 50
2.6.2 State generation . . . 51
2.7 Conclusion . . . 51

3 Lower bounds for the ERCPSP 53
3.1 Introduction . . . 53
3.2 Classical scheduling problems and JPPS . . . 55
3.2.1 m-Machine Scheduling Problem . . . 55
3.2.2 Cumulative Scheduling Problem . . . 55
3.2.3 Generalized Cumulative Scheduling Problem . . . 56
3.2.4 JPPS . . . 56
3.2.4.1 Adaptation of JPPS to CuSP . . . 58
3.2.4.2 Adaptation of JPPS to GCuSP (GJPPS) . . . 59
3.3 Lower bounds for ERCPSP . . . 60
3.3.1 Notation . . . 60
3.3.2 Lower bound based on JPPS . . . 60
3.3.2.1 Transportation problem associated with ERCPSP . . . 61
3.3.2.2 Transformation from ERCPSP into GCuSP . . . 62
3.3.2.3 Improved JPPS Algorithm . . . 63
3.3.3 Extension of Energetic Reasoning . . . 65
3.3.4 Lower bound based on the Shifting Algorithm . . . 66
3.3.4.1 Shifting Algorithm for ERCPSP . . . 66
3.3.5 Lower bound based on Network Flows . . . 68
3.3.5.1 Network Flow problem associated with ERCPSP . . . 68
3.3.6 Method to improve lower bounds . . . 69
3.4 LP-Based lower bound . . . 71
3.4.1 General linear programming scheme . . . 71
3.4.2 Application of the general scheme . . . 72
3.5 Computational results . . . 73
3.6 Conclusion . . . 77

4 Exact Procedures for the ERCPSP **79**
4.1 Introduction . . . 79
4.2 MILP formulations for the ERCPSP . . . 80
4.2.1 Discrete-time formulation (DT) . . . 80
4.2.2 Disaggregated discrete-time formulation (DDT) . . . 81
4.2.3 Flow-based continuous-time formulation . . . 82
4.2.4 Event partitioning based formulation (EP) . . . 83
4.2.5 Computation results . . . 85
4.3 Branch-and-Bound method for ERCPCP . . . 88
4.3.1 Constraint propagation . . . 88
4.3.1.1 Timetabling . . . 89
4.3.1.2 Balance constraint . . . 90
4.3.2 Computation results . . . 92
4.4 Conclusion . . . 93

5 Instance Generation **95**
5.1 Introduction . . . 95
5.2 Basic Concepts . . . 96
5.2.1 Basic Definitions . . . 96
5.2.2 Graph Measures . . . 98
5.3 Basic Data and Precedence Graph Generation . . . 99
5.3.1 Basic Data . . . 99
5.3.2 Precedence Graph . . . 100
5.3.3 Minimal and maximal time lags . . . 103
5.4 Resource Demand and Availability Generation . . . 103
5.4.1 Resource Demand . . . 103
5.4.2 Resource Availability . . . 104
5.5 Hardness of ERCPSP Instances . . . 104
5.6 Functional Description of the generator . . . 107
5.7 Conclusion . . . 109

6 Conclusion and perspectives **111**

References **115**

Liste des figures

2.1 An instance of ERCPSP with seven events and one resource 27
2.2 Resource availability curve 28
2.3 ERCPSP instance with three chains 36
2.4 Three parallel chains 37
2.5 OP-, and OC-subchains of the example of Fig 2.3 38
2.6 The resource level functions 39
2.7 Optimal chain 43
2.8 Series-parallel graph 46
2.9 Example 47
2.10 Interval order graph 48
2.11 Interval representation 48
3.1 The associated ERCPSP 55
3.2 The associated ERCPSP 56
3.3 The associated ERCPSP 56
3.4 Solution of JPPS 57
3.5 Solution of JPPS with the first method 59
3.6 Solution of JPPS with the second method 59
3.7 Solution of GJPPS for an instance of GCuSP 60
3.8 An example: From ERCPSP to GCuSP 64
3.9 Proof of Proposition 3.2 65
3.10 Applying the shifting algorithm 68
3.11 The associated network flow 69
5.1 Effects of the restrictiveness RT on problem hardness 105
5.2 Effects of the resource factor RF on problem hardness 106
5.3 Effects of the resource strength RS on problem hardness 107
5.4 Genrator menu control 107
5.5 Basic data parameters 108
5.6 Acyclic graph parameters 108
5.7 Cyclic graph parameters 108
5.8 Ressource parameters 109

Liste des tableaux

2.1 The resource level functions of the example of Fig 2.3 37
3.1 An instance of the m-machine scheduling problem where m=2 55
3.2 An instance of CuSP 56
3.3 An instance of GCuSP such that $u_1 = 0$, $u_2 = 3$, $b_1 = 6$ and $b_2 = -3$ 56
3.4 Neumann and Schwindt benchmark details 73
3.5 Results of the bounds on the benchmark of Neumann and Schwindt . . 75
3.6 Results of the bounds on J30 76
3.7 Percentage of infeasible instances found 76
3.8 Results of the LP-based lower bounds 77
4.1 Linear relaxation results. 86
4.2 Results of the branch-and-bound method 92
5.1 Constant parameter values 104
5.2 Variable parameter values 104
5.3 Effects of number of events n on problem hardness 105

Liste des algorithmes

1.1 The shifting algorithm 16

2.1 Computation of the earliest start time schedule for a complete linear order. 31
2.2 Algorithm of Johnson 34
2.3 Construction of OP-subchains 40
2.4 construction of OC-subchains 40
2.5 Algorithm to construction an optimal schedule 43
2.6 Algorithm to solve the interval order case 50

3.1 Improved JPPS algorithm. 63
3.2 The shifting algorithm 67
3.3 Destructive bound based on network flows 69
3.4 Algorithm to improve LB 70

5.1 Direct method 100
5.2 Contraction method 101

Notations

Scheduling problem with renewable resource

- Y : set of activities.
- n : number of activities.
- S_i : starting time of activity i.
- p_i : processing time of activity i.
- r_i : release date of activity i.
- q_i : tail of activity i.
- R : set of renewable resources.
- R_ρ : the global resource amount of resource $\rho \in R$.
- E : set of precedence graph.
- e_i^ρ : the requirement for resource ρ by activity i during its execution.

ERCPSP

- X : set of events.
- n : number of events.
- X_k^p : set of production events of resource k.
- X_k^c : set of consumption events of resource k.
- $S(i)$: the occurrence time of event i (also denoted S_i).
- S_i : the occurrence time of event i (also denoted $S(i)$).
- $X(S,t)$: the set of events which have occurred by time t.

- $ES(i)$: the earliest occurrence time of event i (also denoted ES_i).
- ES_i : the earliest occurrence time of event i (also denoted $ES(i)$).
- K : set of nonrenewable resources.
- Q_k : the initial level of resource $k \in K$.
- a_i^k : the quantity of resource k produced or consumed by event i.
- U : set of generalized precedence graph.
- v_{ij} : the time lag between events i and j.
- $l_{i,j}$: the length of the longest path from i to j.
- $\Gamma^+(i)$: the set of direct successors of event i.
- $\bar{\Gamma}^+(i)$: the set of ascendants of event i.
- $\Gamma^-(i)$: the set of direct predecessors of event i
- $\bar{\Gamma}^-(i)$: the set of descendants of event i.
- H : the set the scheduling horizon.

The single-resource case

- X^p : set of production events.
- X^c : set of consumption events.
- a_i : the number of resource units produced or consumed by event i.

Parallel chain case

- SP : the number of parallel chains.
- L_h : the number of events of chain h.
- C_i^h : the ith event of chain h.
- $b_h(i)$: the number of resource units produced or consumed by event C_i^h.
- $E_h(i)$: the earliest occurrence time of event C_i^h.
- $A_h(i)$: the level of resource at the earliest occurrence time of event C_i^h.

- $A(e|S)$: the level of resource at the occurrence time of event $e \in X$.
- $m(S)$: The minimal level of resource associated with S.

Introduction

The concept of scheduling is not new. Sun Tzu wrote about scheduling and strategy from a military point of view 2500 years ago. The pyramids were built 3000 years ago and transcontinental railways were constructed for some 200 years. All these achievements could not be accomplished without some kind of schedule. At the beginning of this century, scheduling began to be taken seriously in manufacturing with the work of Henry Gantt and other pioneers. The scheduling theory is younger. Indeed, it took many years for the first scheduling publications to appear in operations research literature. Some of the first publications came in the early fifties in Naval Research Logistics Quarterly and involved results by [Johnson, 1954], [Smith, 1956] and [Jackson, 1956]. In the sixties, an important number of works was done on dynamic programming and integer programming formulations of scheduling problems. In the seventies, the research focused mainly on the complexity hierarchy of scheduling problems. Since the eighties, different directions in industry and academia have been studied with an increasing amount of attention devoted to stochastic scheduling problems.

Scheduling problems are defined by activities (tasks) or events (milestones) that have to be performed in accordance with a set of precedence and resource constraints. Each activity has a duration and normally requires resources. An event (milestone) refers to a stage of accomplishment associated with a certain point in time. Resources may be renewable or nonrenewable. Renewable resources are available each period without being depleted. Typical examples of renewable resources include manpower, machines, tools, equipment, space, etc. Nonrenewable resources are depleted as they are used. The money is the best example of nonrenewable resource for which Carlier and Rinnooy Kan introduced the financing problem [Carlier and Rinnooy Kan, 1982]. The application fields of scheduling theory include computers, manufacturing, agriculture, hospitals, transport, etc. The principal focus is on the optimal allocation of one or more resources to activities over time [Lawler et al., 1993, Lee et al., 1997, Brucker, 2007, Pinedo, 2012].

The Resource Constrained Project Scheduling Problem (RCPSP) is without doubt the most widely studied scheduling problem in literature. In this problem, non-preemptive activities requiring renewable resources, and subject to

precedence constraints, have to be scheduled within a minimal makespan. The RCPSP with minimum and maximum time lags has also been the subject of several papers [Bartusch et al., 1988] [Cesta et al., 2002] [Neumann and Zhan, 1997] [Neumann et al., 2006], it also concerns only renewable resources. In this thesis, we address the Extended Resource Constrained Project Scheduling Problem (ERCPSP). ERCPSP is a general scheduling problem where the availability of resources is depleted and replenished [Carlier et al., 2009, Carlier et al., 2016]. An instance of ERCPSP consists of events, nonrenewable resources and generalized precedence constraints between pairs of events. Each event produces or consumes some units of resources at its occurrence time. The objective is to build a schedule that satisfies the precedence and resource constraints and minimizes the makespan. ERCPSP is a generalization of RCPSP where activities requiring renewable resources are replaced by events consuming or producing nonrenewable resources.

Some other authors have worked on models similar to ERCPSP. We can quote the works of [Neumann and Schwindt, 2002] and of [Laborie, 2002]. Neumann and Schwindt formalized the Project Scheduling Problem with Inventory Constraints where the availability of each resource is at any time upper and lower bounded. To solve this problem, they proposed a branch-and-bound algorithm with a filtered beam search heuristic. Laborie introduced the concept of a Resource Temporal Network (RTN). He proposed a constraint propagation algorithm to solve the problem. Koné et al. worked on the RCPSP with Consumption and Production of Resources (RCPSP/CPR) [Koné et al., 2013]. The particularity of their extension of RCPSP is that, in addition to renewable resources considered in the basic version, it also involves nonrenewable resources which can be consumed (or not) at the starting time of an activity in a certain amount and/or then produced in another amount at the completion time of this activity. To solve this problem, Koné et al. proposed four mixed integer linear programming models for RCPSP/CPR. ERCPSP coincides with the problem considered by [Neumann and Schwindt, 2002] and [Laborie, 2002] where no upper bound on the resource availability is prescribed.

We are interested in this thesis in proposing new methodologies and approaches to solve ERCPSP. Chapter 1 is concerned with giving the reader a background on scheduling problems and several methods previously proposed to solve them. More precisely, we present first some definition and terminology dedicated to scheduling problems. Then, we introduce the RCPSP and we present some lower bounds, exact procedures and heuristics to solve it. Finally, we study three scheduling problems with nonrenewable resources. The first problem is the financing problem which can be solved using the shifting algorithm, the second problem is the Project Scheduling Problem with Inventory Constraints of [Neumann and Schwindt, 2002] and the last

one is the RCPSP with consumption and production of resources.

Following the presentation of the field of the study of the thesis, we present in Chapter 2 some terminology dedicated to basic concepts and formulate the ERCPSP, and we show the connection between this problem and other scheduling problems with production and consumption of resources. Then, we study four special cases of ERCPSP for which the decision problem can be solved in polynomial time: the relocation problem, the parallel chain case, the series-parallel case and the interval order case. An adaptation of the algorithm of [Abdel-wahab and Kameda, 1978] is proposed for the parallel chain case. This algorithm is based on a decomposition of chains into production and consumption subchains. These subchains can be seen as jobs of a flow-shop with two machines. The idea is to construct a schedule of standard form, where the events of each subchain are scheduled next to each other, from any feasible schedule. Then, the Johnson's rule is applied to these subchains in order to obtain an optimal sequence. A list algorithm is introduced for the interval order case to construct feasible schedules. The priorities of events are defined using the proprieties of interval orders. Finally, a dynamic programing algorithm is proposed to solve the parallel chain case.

In Chapter 3, we introduce six lower bounds for ERCPSP. Two of them are based on the extraction of a generalized Cumulative Scheduling Problem, combined with an adapted version of Jackson's Pseudo-Preemptive Schedule [Carlier and Pinson, 2004] and the concept of energetic reasoning. Two further lower bounds respectively result from applying Carlier and Rinnooy Kan's Shifting Algorithm to a Financing Problem and iteratively testing the feasibility of associated network flow problems in a dichotomic search method. The last two lower bounds are destructive lower bounds computed using a general linear programming scheme. This linear programming scheme is based on a decomposition of the time horizon into successive intervals.

In the continuity of Chapter 3, Chapter 4 deals with the exact solving of ERCPSP. In the first half of the chapter, we present four mixed integer linear programming formulations for ERCPSP. More precisely, we propose first an adaptation of two known time-indexed MILP formulations of RCPSP to ERCPSP. Second, we introduce an adaptation of a flow-based continuous-time formulation. Finally, we propose a new MILP formulation based on the concept of event partitioning to solve the problem. The time-indexed formulations involve pseudo-polynomial numbers of constraints and variables, since the number of binary variables increases proportionally with the time horizon. However, the two other formulations involve polynomial numbers of constraints and variables. In the second half of the chapter, we present a branch-and-bound method to solve the ERCPSP. The branching process of this

method is similar to the one proposed by [Demeulemeester and Herroelen, 1990] for the RCPSP. The proposed search tree is a binary tree. Each node represents a subset of solutions which satisfy a set of precedence constraints, and it is evaluated using our lower bounds. Finally, we present two adapted constraint propagation algorithms to reduce the number of nodes: the timetabling of [Le Pape, 1994] and the balance constraint of [Laborie, 2002].

In Chapter 5, we develop an instance generator for ERCPSP based on the methodology of ProGen [Kolisch et al., 1995]. This generator takes into account several graph measures such as the number of nodes, the graph complexity, the number of predecessors and successors of a node as well as parameters for the generation of the basic data and the resource constraints.

Finally, the different parts of the thesis are discussed in a general conclusion and future works and perspectives are presented.

Scheduling Problems

Sommaire

1.1 Introduction **5**
1.2 Definition of Scheduling Problems **6**
1.3 Resource Constrained Project Scheduling Problem . . . **8**
1.4 Scheduling problems with non-renewable resources . . . **15**
1.5 Conclusion **20**

1.1 Introduction

Scheduling has been the subject of extensive research since the early 1950s. The application fields of scheduling theory include computers, manufacturing, agriculture, hospitals, transport, etc. The principal focus is on the optimal allocation of one or more resources to activities over time. (see [Lawler et al., 1993, Lee et al., 1997, Brucker, 2007] and [Pinedo, 2012]).

Scheduling problems may be polynomially solvable or NP-hard. The NP-hardness of a problem means that it is impossible to obtain an optimal solution without using an enumerative algorithm, for which computation times increase exponentially with problem size. To solve NP-hard scheduling problems, a branch-and-bound algorithm is usually applied. In practice it is better to use a heuristic method to find quickly an approximate solution.

More recently the Resource Constrained Project Scheduling Problem (RCPSP) has been extensively studied. The RCPSP and its extensions are very general scheduling models which contain all complex machine scheduling problems as special cases [Brucker et al., 1999]. In RCPSP, non-preemptive activities requiring renewable resources, and subject to precedence constraints, have to be scheduled in order to minimize makespan. Renewable resources are allocated to activities at their starting time and released at their completion time. On the contrary, a *nonrenewable*

resource is produced or consumed by an activity only at its starting time. Money is an example of nonrenewable resource for which [Carlier and Rinnooy Kan, 1982] introduced the financing problem.

The remaining of this chapter is structured as follows. In Section 1.2 we present some definition and terminology dedicated to scheduling problems. In Section 1.3 we present the RCPSP which is the most famous scheduling problem with renewable resources. In Section 1.4 we present three scheduling problems with nonrenewable resources, and finally we conclude the chapter in Section 1.5.

1.2 Definition of Scheduling Problems

A scheduling problem consists of a number of activities (tasks) or events (milestones) that have to be performed in accordance with a set of precedence and resource constraints. Each activity has a duration and normally requires resources. An event (milestone) refers to a stage of accomplishment associated with a certain point in time. Resources can be of different types, including manpower, machinery, financial resources, energy, etc. The finish-start relation with a zero time lag is the usual type of precedence relation. An activity can only occur as soon as all its predecessor activities have occurred. Other precedence relations exist such as start-start, finish-finish and start-finish relations, with various types of minimal and/or maximal time lags.

1.2.1 Activities and Events

Activities and events are the basic entities in scheduling problems. An activity is characterized by a starting time S_i, a completion time C_i and a processing time p_i. An activity has specific requirements on the amounts and types of resources. The set of activities is denoted $Y = \{0, 1, ..., n+1\}$. By convention, the two activities 0 and $n+1$ are added to respectively define the start and the end of the schedule. Activities may be non-preemptive or preemptive. A non-preemptive activity is executed without interruption from its starting time to its completion time. Preemptive activity can be interrupted at any time. According to the scheduling problem, an activity i may also have the following characteristics:

- A release date (r_i) is the time when activity i is available to start processing.
- A tail (q_i) is the latency between the completion of activity i and the completion of the project.

- A due date (resp. deadline) (d_i) denotes the latest time instant at which activity i has to finish.

- A weight (w_i) signifies the importance of activity i.

An event is characterized by an occurrence time S_i and has no processing time. The set of events is denoted $X = \{0, 1, ..., n+1\}$. The two events 0 and $n+1$ are added to respectively define the start and the end of the schedule. An event can produce or consume a quantity of resource at its occurrence time.

1.2.2 Resources

Activities and events require resources for their execution. Each resource has a limited capacity and may be renewable or nonrenewable. Renewable resources are available each period without being depleted. The set of renewable resources is denoted by R. The constant availability of renewable resource $k \in R$ is denoted R_k. Typical examples of renewable resources include manpower, machines, tools, equipment, space, etc. Nonrenewable resources are depleted as they are used. The money is the best example of nonrenewable resources. The set of nonrenewable resources is denoted by K. The availability of nonrenewable resource $k \in K$ is denoted Q_k.

1.2.3 Precedence relations

Precedence relations between activities can be expressed by linear constraints between the starting times and the completion times of activities. For example a finish-start precedence relation with zero time lag between activities i and j is modeled by the linear constraint $S_j \geq C_i$. This relation implies that activity j can start immediately after the completion time of activity i, or later. Other type of precedence relations exist such as start-start relation ($S_j \geq S_i$: activity j can only start after the starting time of activity i), start-finish relation ($C_j \geq S_i$: activity j can only finish after the starting time of activity i) and finish-finish relation ($C_j \geq C_i$: activity j can only finish after the completion time of activity i). Note that we have only one type of precedence relations between events and it is the start-start relation.

Each precedence relation (i, j) has a time lag v_{ij} to denote the latency between i and j. Time lags can be positive, zero or negative. A finish-start relation with a non-zero time lag $v_{ij} \neq 0$ implies that activity j can only start after $C_i + v_{ij}$. When v_{ij} is negative (resp. positive), we talk about maximal (resp. minimal) time lag.

1.2.4 Constraints and Objectives

Constraints and objectives are defined during the formulation of the problem. Constraints define the feasibility of a schedule. Objectives define the optimality of a schedule. Constraints must be satisfied and objectives should be optimized. Both constraints and objectives may be resource-based, activity-based or a combination of them. Constraints appear in many forms. Precedence constraints define the order in which activities can be performed. Resource constraints define the requirements of resources.

A feasible schedule satisfies all of the constraints. An optimal schedule not only satisfies all of the constraints, but also is better than any other feasible schedule. Goodness is defined by objective function. Typical example of objectives include minimization of the makespan, minimization of cost, maximization of resource utilization, resource efficiency, minimization of work-in-progress, etc.

1.3 Resource Constrained Project Scheduling Problem

The Resource Constrained Project Scheduling Problem (RCPSP) is the basic problem type in project scheduling. It is without doubt the most widely studied scheduling problem with renewable resources in literature, and it has resulted in an overwhelming amount of papers with solution procedures devoted to it. In this problem, non-preemptive activities requiring renewable resources, and subject to precedence constraints, have to be scheduled in order to minimize the makespan.

1.3.1 Problem Description

An instance I of the Resource Constrained Project Scheduling Problem (RCPSP) is defined by:

- A set Y of n activities: $\forall i \in Y$, p_i denotes the processing time of i.
- A set E of precedence relations: $(i, j) \in E$ means that activity i precedes activity j (j cannot start before i is over).
- A set R of renewable resources: $\forall k \in R$, R_k denotes the global resource amount of resource k and $\forall i \in Y, \forall k \in R$, e_i^k denotes the requirement for resource k by activity i during its execution. These resources are given back to the system once the activity is completed. The total resource demand must be less

than or equal to the global resource amount for each resource throughout the scheduling horizon.

A schedule $\mathcal{S}$ is a function assigning a starting time S_i to each activity $i \in Y$. It is feasible if it satisfies the precedence and resource constraints. Thus, solving RCPSP means computing a feasible schedule $\mathcal{S}$ with a minimal makespan.

1.3.2 Lower bound calculations

In this section we present methods for computing lower bounds for the RCPSP. Lower bounds are used to estimate the quality of heuristic solutions. They are also needed for the construction of branch-and-bound algorithms.

1.3.2.1 Critical Path and Capacity Bounds

The Critical Path Bound (lb_0) is the most obvious and the most frequently used lower bound. This bound is based on ignoring the resource constraints. So, it is obtained by computing the length of a critical path in the project network. Another simple bound, called Capacity Bound (lb_1), is obtained by focusing on the resource constraints instead of the precedence constraints, it is computed as follows:

$$lb_1 = \max_{k \in R} \lceil \sum_{i \in Y} \frac{e_i^k \times p_i}{R_k} \rceil$$

[Stinson et al., 1978] introduced the critical sequence lower bound (lb_2), which takes into account simultaneously precedence and resource constraints. Let us consider a critical path CP in the graph (Y, E). For each activity $i \in Y - CP$, it is determined how many time periods p'_i activity i can be scheduled in parallel to the critical path (taking into account the resource constraints). In case of $p'_i < p_i$, activity i cannot be processed completely and the project cannot finish before $lb_0 + p_i - p'_i$. Therefore, lb_2 is defined as follows:

$$lb_2 = lb_0 + \max\{0, \{p_i - p'_i | i \in Y - CP\}\}$$

1.3.2.2 Bin Packing Bounds

RCPSP can be relaxed by ignoring the precedence constraints and all resource constraints but one for some resource k. The obtained problem is similar to single resource constrained problems of the bin packing type. Thus, lower bounds for such problems could be generalized for RCPSP, see for instance [Berger et al., 1992, Scholl et al., 1997].

1.3.2.3 Node Packing Bounds

These bounds consist of finding a subset SY of activities, such that each activity $i \in SY$ is incompatible with any other activity within this set. So, the activities of SY must be scheduled sequentially and the sum of their processing times represents a lower bound for RCPSP. The problem of determining a set SY which maximizes the obtained lower bound value can be formulated as a weighted-node packing problem [Mingozzi et al., 1998].

1.3.2.4 Parallel Machine Bounds

These lower bounds are based on m-machine scheduling problems generated from RCPSP. The m-machine scheduling problem is defined as follows. A set Y of activities have to be executed on m identical machines. At most one activity is executed on one machine at a time. A basic lower bound for this problem is the quantity $G'(Y)$ defined as [Carlier, 1984]:

$$G'(Y) = (1/m) \times (r_{i_1} + ... + r_{i_m} + \sum_{i \in Y} p_i + (q_{i_1} + ... + q_{i_m})$$

The bound $G'(Y)$ can be improved using the Jackson's Pseudo Preemptive Schedule [Carlier and Pinson, 1998], which is an efficient lower bound for the m-machine problem.

1.3.2.5 Precedence Bounds

These lower bounds are based on incompatible pair of activities. For each incompatible pair (i, j), a disjunctive precedence relation can be introduced, because i must be finished before j can be started or vice versa. Therefore, lower bounds can be computed by testing both directions of the disjunctive precedence relation [Balas, 1968]. These lower bounds can be improved by considering incompatible triplets of activities.

1.3.2.6 LP-Based Bounds

Authors of [Mingozzi et al., 1998, Brucker and Knust, 2000] proposed lower bounds based on linear programming formulations. They relaxed the non-preemption constraints and associate a variable with each subset of activities that can be executed simultaneously without violating any constraint. As a result the linear program can be very large, however it can be tackled by column generation. A general linear programming scheme for computing new bounds for RCPSP was proposed

by [Carlier and Néron, 2003]. It is based on a decomposition of the time horizon into several successive intervals. The interest of this scheme was demonstrated theoretically.

1.3.3 Exact procedures

One of the most commonly used methods for solving RCPSP is the branch-and-bound method. Mixed Integer Linear Programming (MILP) is used to model RCPSP. It can be solved either using branch-and-price, branch-and-cut or some other method [Brčić et al., 2012].

1.3.3.1 Linear programming based approaches

There are a large number of MILP formulations for the RCPSP. Among others, we can quote the formulations involving an exponential number of variables such as the discrete time formulation of [Mingozzi et al., 1998] which considers that all feasible sets of activities (all activities included in such a set can be processed simultaneously) of the problem are known, and the continuous time formulation of [Alvarez-Valdés and Goerlich, 1993] which assumes that all forbidden sets (distinct sets whose elements are activities which cannot be processed simultaneously) are given. On the other hand, there exist formulations involving a pseudo-polynomial number of variables (such as the formulation of [Christofides et al., 1987] involving time-indexed variables), and other formulations involving a polynomial number of variables (such as the formulation of [Artigues et al., 2003] involving sequencing variables). Authors of [Koné et al., 2011] proposed an efficient MILP formulation for RCPSP involving event variables. This formulation is also used as modeling inspiration for other solving methods.

1.3.3.2 Branch-and-bound procedures

The branch-and-bound method is a well known technique for solving combinatorial problems. This method was described by [Agin, 1966], it allows the generation of optimal solutions within an acceptable computational time. The RCPSP belongs to the set of complex combinatorial problems. A combinatorial problem refers to the assignment of numerical values to a finite set of variables, in order to satisfy a set of constraints and to minimise an objective function $f(x)$. The constraints of combinatorial problems may be implicit or explicit. Implicit constraints are satisfied by the way in which the branch-and-bound algorithm is constructed (example: precedence constraints). Explicit constraints, however, need procedures for recognition as an integral part of the branch-and-bound algorithm (example:

resource constraints). A solution s is an assignment of numerical values to a set of variables that satisfies all the implicit constraints. In RCPSP, an earliest starting time schedule represents a solution. A feasible solution is defined to be an assignment of numerical values that satisfies all the constraints (implicit and explicit ones). A solution in which the activities are processed one after the other respecting all precedence relations represents a feasible solution.

Let Ω denote the set of all solutions of a combinatorial problem. A partition of Ω is a collection of subsets $\Omega_1, \Omega_2, ..., \Omega_s$ with the following properties:

$$\Omega_1 \cup \Omega_2 \cup ... \cup \Omega_s = \Omega$$

$$\Omega_i \cap \Omega_j = \varnothing, \quad \forall i \neq j$$

The branching process starts with partitioning the set Ω into several subsets $\Omega_1, ..., \Omega_s$. By continuously partitioning such subsets again a search tree with subsets of solutions as nodes is constructed. The initial node of the tree is the set of all solutions Ω. Branches are created by the branching process and nodes of the tree represent the subsets Ω_i of Ω. With each node i, a subset Ω_i is associated. The branching process stops, if the optimal solution of each leaf node of the tree is known or can efficiently be computed. A branching scheme defines how the initial set Ω as well as the subsets Ω_i are partitioned. A search strategy defines the sequence in which the nodes of the search tree are considered. In general, we distinguish two search strategies. The first one is a backtracking strategy that selects an intermediate node which was created at the previous stage. The second one is a best-first strategy that selects the intermediate node which has the best lower bound.

Two characteristics of the branch-and-bound algorithm are necessary in order to solve the combinatorial problem optimally. The first characteristic is called the branching characteristic, which ensures the generation of an optimal solution during the branching process. The second characteristic, which is called the bounding characteristic, gives the possibility to identify an optimal solution. For example, if during the search process a feasible solution s', whose objective value $f(s')$ is smaller than or equal to the lower bounds lb_i of each intermediate node i, is obtained then s' is an optimal solution and no further branching from the nodes i are needed. Using the definitions given above, a branch-and-bound algorithm could be defined as follows:

1. branching from end nodes to new nodes: given an intermediate node, we create new nodes,

2. determining lower bounds for the new nodes,

3. choosing an intermediate node from which to branch next,
4. recognizing when a node contains only infeasible or non-optimal solutions,
5. recognizing when a final node contains an optimal solution.

In the sequel of this section, different branch-and-bound procedures will be presented. Note that the list of the presented branching schemes is not exhaustive.

1.3.3.3 Chronological branching scheme

An easy way to build solution is to associate a partial schedule with each node of the search tree, and the branching scheme consists of adding at least one activity to this partial schedule. Thus, a leaf of such a tree corresponds to a feasible schedule. There are two families of chronological branching schemes. The first one considers that at most one activity is added to the partial solution at each node of the tree, whereas in the second one feasible subsets of activities are added to the partial schedule.

Adding one activity to a partial solution: Let N be a given node of the search tree that corresponds to a partial schedule. The set of eligible activities (EL) is the set of activities whose all predecessors have been already scheduled. The most natural method to extend a partial solution is to consider this set of eligible activities EL. A new node is created for each activity in EL and this activity is scheduled as soon as possible, at a time-point greater than the starting time of the activity scheduled at the previous level and satisfying both resource and precedence constraints, see for instance [Sprecher, 2000].
[Baptiste et al., 1999] introduced another method to extend a partial solution. An activity $i \in EL$ is chosen (the one with minimum earliest starting time) and two new nodes are created. In the first node activity i is scheduled as soon as possible. In the second node, at least one activity in $EL \setminus \{i\}$ is enforced to start before or simultaneously with the start of i.

Delaying alternatives: This branching scheme was initially introduced by [Christofides et al., 1987]. Improvements and corrections of this method was proposed by [Erik L. Demeulemeester, 1997]. Let us consider a given node that corresponds to a partial schedule, and a time t which corresponds to the completion time of an activity scheduled at a previous level. We define the set $IP(t)$ (resp. $EL(t)$) of in progress activities (resp. eligible activities) at time t. We consider that an activity is eligible if all its predecessors are completed at t. These eligible activities are added to the partial schedule at time t. If no resource conflict occurs, then t is increased, otherwise all Minimal

Delaying Alternatives are enumerated and a node is created for each one of them. A minimal delaying alternative (MDA) is a subset of $IP(t) \cup EL(t)$, such that if the activities of this subset are delayed then the resource conflict disappears. Each subset of a MDA is not a delaying alternative.

1.3.3.4 Minimal forbidden sets

This branching scheme was introduced by [Lgelmund and Radermacher, 1983] and is totally different from the previous ones. A forbidden set F is a set of activities that violate the resource constraints if they are performed concurrently. A forbidden set F is called minimal forbidden set if it does not admit any forbidden set as a subset. Suppose that a resource conflict is detected at a time-point t. This resource conflict corresponds to at least one forbidden set F. So, to solve it, we add precedence constraints between any two activities of F in order to delay one or more than one activity. Theoretical aspects related to forbidden set enumeration for Branch-and-Bound was investigated by [Stork and Uetz, 2005].

1.3.4 Heuristic procedures

Heuristic procedures dominate the research on RCPSP. The simplest heuristics are constructive ones that use a priority list [Brčić et al., 2012]. The basic members of this family are serial and parallel scheduling schemes. These methods are very fast, even for huge projects but they do not produce good solutions. In a recent paper, [Valls et al., 2005] showed that incorporating a technique called double justification (DJ) in RCPSP heuristic algorithms can produce a substantial improvement in the results obtained.

Metaheuristics are improvement heuristics which aim to produce solutions reasonably close to the optimum. Decomposition based genetic algorithm for RCPSP was introduced in [Debels and Vanhoucke, 2007], This metaheuristic yields some best results on standard benchmarks. Hybrid algorithm using a combination of scatter search and ant colony optimization was used by [Chen et al., 2010] to get very good results on RCPSP. Other methods were used such as: taboo search, simulated annealing, electromagnetism metaheuristics, etc. Authors of [Kolisch and Hartmann, 2006] concluded that the best metaheuristics are the ones that use population-based approaches.

1.4 Scheduling problems with non-renewable resources

In this section, we present three scheduling problems with non-renewable resources. The first one is the Project Scheduling Problem with Inventory Constraints, which was defined by Neumann and Schwindt [Neumann and Schwindt, 2002]. The second problem is the Financing problem and the last one is the RCPSP with consumption and production of resources.

1.4.1 The Financing Problem

The Financing Problem [Carlier and Rinnooy Kan, 1982, Slowiński, 1984] has been investigated prior to the Project Scheduling Problem with Inventory Constraints. This problem aims to model the financing of some project. It is a special case of ERCPSP, insofar as the dates of production events are given whenever there are precedence constraints between consumption events. It is solved using a polynomial algorithm known as the shifting algorithm. Deadlines can also be taken into account.

1.4.1.1 Problem description

An instance of this problem is defined by a set X of n consumption events. Each event $i \in X$ consumes a_i units of a nonrenewable resource at its occurrence time. Initially, at time $z_1 = 0$, b_1 units of resource are available. An additional quantity of $b_2, ..., b_q$ units of resource becomes available at given times $z_2, ..., z_q$. A precedence graph $G = (X, E)$ is associated with the problem. X contains the set of consumption events and the two fictitious events 0 and $n + 1$. We suppose that the minimum capacity is 0 and the maximum capacity is infinite. When money is involved, it is better to have as large a stock as possible! At first we will suppose that G does not contain any arc $(i, 0)$. Such an arc will model a deadline. All events have to be scheduled in a minimal makespan. This problem can also be modeled using ERCPSP.

1.4.1.2 Feasible Schedule

Let us denote π_{ij} the value of a maximal path from i to j in G. $ES = \{ES_i = \pi_{0i}/i \in X\}$ is the earliest time-feasible schedule and $LS = \{LS_i = \pi_{0,n+1} - \pi_{i,n+1}/i \in X\}$ the latest time-feasible schedule. Let $LS(\delta) = \{LS_i + \delta/i \in X\}$ be a latest schedule with makespan equal to $\pi_{0,n+1} + \delta$ (δ is positive). A time-feasible schedule $S = \{S_i/i \in X\}$ is resource-feasible if the following condition is satisfied: for any $t : R(t) = \sum_{\{i/S_i \leq t\}} a_i \leq A(t) = \sum_{\{\mu \in [1...q]/\tau_\mu \leq t\}} b_\mu$. In other words, the requirement

curve is below the availability curve.

1.4.1.3 The Shifting Algorithm

The financing problem can be solved using the shifting algorithm in polynomial time ($O(n \log n)$). In the shifting algorithm, the latest schedule is shifted in order to satisfy the feasibility condition. Calculation details are given in Algorithm 1.1.

Algorithm 1.1: The shifting algorithm

```
begin
    if (∑_{i∈X} a_i > ∑_{μ∈[1,...,q]} b_μ) then
        there is no feasible schedule
    else
        Compute the latest schedule LS := {LS_i = π_{0,n+1} − π_{i,n+1}/i ∈ X};
        A(τ_1) := b_1;
        for μ := 2 to q do
            A(τ_μ) := A(τ_{μ−1}) + b_μ
        μ := 1; δ := 0; R := 0;
        for i := 1 to n do
            R := R + a_i;
            while A(τ_μ) < R do
                μ := μ + 1
            if δ < (τ_μ − LS_i) then
                δ := (τ_μ − LS_i)
```

1.4.2 The Project Scheduling Problem with Inventory Constraints

The Project Scheduling Problem with Inventory Constraints was introduced by Neumann and Schwindt [Neumann and Schwindt, 2002]. Inventory constraints refer to nonrenewable resources, which can be stocked and have minimum and maximum prescribed stocks. Neumann and Schwindt proposed a branch and bound method for solving this problem and truncated it to a filtered beam search heuristic.

1.4.2.1 Problem description

An instance of the Project Scheduling Problem with Inventory Constraints is defined by a set X of events, a set K of nonrenewable resources, and a set U of generalized precedence relations. For each event $i \in X$ and for each resource $k \in K$, a_i^k denotes the quantity of resource k produced or consumed by event i. For each $(i, j) \in U$, v_{ij}

represents the time lag between events i and j. The length of the longest path from i to j in the graph (X, U) is denoted π_{ij}. Inventory constraints refer to nonrenewable resources, which can be stocked and have minimum and maximum prescribed stocks. So, a safety stock $\underline{Q_k}$ and a capacity of the storage facility $\overline{Q_k}$ are given for each resource $k \in K$. A schedule S is a function giving an occurrence time S_i to each event $i \in X$. It is time-feasible if it satisfies all the precedence constraints. It is resource-feasible if, for each resource k and for every time t, the inventory of the resource is between $\underline{Q_k}$ and $\overline{Q_k}$. A schedule is said to be feasible if it is both time-feasible and resource-feasible. The objective in this problem is to find a feasible schedule that minimizes the makespan.

1.4.2.2 The Exact Method of Neumann and Schwindt

[Neumann and Schwindt, 2002] propose a branch and bound method which enumerates alternatives for avoiding stock shortages and surpluses by introducing *disjunctive precedence constraints* between some disjoint sets of events A and B: A is before B if $min\{S_i/i \in B\} \geq min\{S_i/i \in A\}$. So a node in the tree corresponds to a set of these disjunctive precedence constraints. The authors show that if the set of schedules of a node is not empty, there exists an earliest schedule S respecting the initial precedence constraints and the disjunctive precedence constraints. Moreover they explain how to compute this earliest schedule by adjusting the starting times of events. If this schedule is resource-feasible, it is feasible, so we backtrack. Otherwise, there exists a time t such that $X(S,t) = \{i \in X \mid S_i \leq t\}$ is a surplus set or a shortage set. That is: $\sum_{i \in X(S,t)} a_i^k > \overline{Q_k}$ or $\sum_{i \in X(S,t)} a_i^k < \underline{Q_k}$. They consider the smallest t satisfying the previous condition.

Let us explain the case of a surplus set. Here it is necessary to postpone a minimal subset B of events (B included in $X(S,t)$). B is composed of events i with a_i^k strictly positive (production events) and it is minimal in the sense that the surplus conflict is solved, but it is not solved for some proper subset of B. Of course it is necessary to enumerate several alternatives for B. To postpone B, they consider the set A of events which are not in $X(S,t)$ and with a_i^k strictly negative (consumption events).

Neumann and Schwindt proposed two lower bounds. The first lower bound is associated with the earliest schedule. The second lower bound uses the earliest schedule ES ($\forall i \in X, ES_i = \pi_{0i}$) and also a latest schedule LS ($\forall i \in X, LS_i = C_{max} - \pi_{i,n+1}$ where C_{max} is a hypothetic makespan). A lower bound of the stock is obtained by starting the consumption events as early as possible, given by ES, and the production events as late as possible, given by LS. Similarly an upper bound of the stock is obtained by starting the consumption events as late as possible, given

by LS, and the production events as early as possible, given by ES. If at some time t it can be proved that the stock will be insufficient or too large, there is no solution and the corresponding node of the tree can be cancelled.

The experimental analysis of [Neumann and Schwindt, 2002] shows that their method can solve problem instances with 100 events and five storage resources. These data were generated by the authors, and are composed of 360 projects. Twelve projects have not been solved optimally. The larger problems are solved by replacing the minimal subset B by a subset of cardinality one.

1.4.2.3 The Exact Method of Laborie

The scheduling problems which are considered by [Laborie, 2002] are very general. They include the Resource Constrained Project Scheduling Problem and the project scheduling problem with inventory constraints. His method propagates resource constraints within a constraint programming approach. Its application to the project scheduling problem with inventory constraints is presented below. [Laborie, 2002] refers to a resource reservoir, because the maximum and minimum capacities are given. Most of the techniques in the literature refine the execution intervals of activities. These are the cases of edge finding or energy based reasoning devoted to renewable resources. But generally, at the start of the search, no activity intervals can yet be deduced. So [Laborie, 2002] focuses on the precedence relations between events rather on their absolute positions in time, as we explain below. It is a method that is complementary to the aforementioned techniques.

The search space consists of a global search tree. The method consists in iteratively refining a partial schedule. A partial schedule is composed of a set of events, temporal constraints and resource constraints. The main tools of [Laborie, 2002] are time-tabling, the resource graph and the balance constraints.

Time-tabling is a propagation technique which relies on the computation of upper and lower bounds at any time t for the use of every resource k. It can limit the domains of the start and completion times of activities by removing the dates that would necessarily lead to an over-consumption or under-consumption of some resource by an event.

The resource graph RG is composed of two sets of arcs: $RG = (V, E_{\leq}, E_{<})$, where $E_{<}$ is included in $E_{\leq}$, and

- $E_{\leq}$ is the set of couples (i, j) such that: $S_i \leq S_j$,
- $E_{<}$ is the set of couples (i, j) such that: $S_i < S_j$.

The resource graph expresses precedence relations between events. The graph on a resource is designed to gather together all the precedence relations between events

on the resource. They may come from initial temporal constraints, from deductions and from branching decisions. When new precedence relations are introduced, the transitive closure of the resource graph is maintained thanks to a matrix.

This graph means that balance constraints associated with events can be evaluated. The basic idea is to compute, for each event using a specific resource, upper and lower bounds on the resource level just before and just after this event. An event i is safe for some resource if the upper bound of the resource level just before i and just after i is smaller than the maximum capacity, and the lower bound of the resource level before i and after i is larger than the minimum capacity. When all events of a resource are safe, the reservoir constraint of this resource is satisfied.

The balance constraint can reveal three types of information: dead-ends, new bounds for time variables and new precedence relations. For instance, when the upper bound of the resource level just before event i is strictly smaller than the minimal capacity, we get a dead-end. We can also get new bounds on time variables. For instance, if the resource level before i in the partial schedule is smaller than the minimum stock, production events need to be scheduled before i. The earliest dates can be computed at which sufficient resources might be available for processing i. It will depend on the earliest dates at which production events can be scheduled. Finally if the processing of event j after event i would provoke a dead-end, i must be scheduled before j. So we can add the corresponding precedence constraint.

Branching is based on precedence relations. It involves choosing two events i and j. Laborie chooses either to process i before j or j before i, one of both precedences being strict. i is chosen as a critical event, for instance an event consuming or producing a large quantity of resources. The choice which is sophisticated is explained in the paper.

Laborie's method has been implemented in the ILOG Scheduler, a C++ library for constraint-based scheduling. It can solve to optimality all the instances of [Neumann and Schwindt, 2002], including the twelve previously open instances in less than 10 seconds. To resume this method is elaborated and innovative. It is also very efficient in practice.

1.4.3 RCPSP with Consumption and Production of Resources

The RCPSP with consumption and production of Resources (RCPSP/CPR) was introduced by Koné et al. [Koné et al., 2013]. The particularity of this extension of RCPSP is that, in addition to renewable resources considered in its basic version, it also involves nonrenewable resources. Koné et al. [Koné et al., 2013] proposed four

mixed integer linear programming models to solve the problem. An instance I of RCPSP/CPR is defined by:

- A set Y of n activities: $\forall i \in Y$, p_i denotes the processing time of i.
- A set E of precedence relations: $(i, j) \in E$ means that activity i precedes activity j (j cannot start before i is over).
- A set R of renewable resources: $\forall \rho \in R$, R_ρ denotes the global resource amount of resource ρ and $\forall i \in Y, \forall \rho \in R$, e_i^ρ denotes the requirement for resource ρ by activity i during its execution. These resources are given back to the system once the activity is over. The total resource demand must be less than or equal to the global resource amount for each resource throughout the scheduling horizon.
- A set K of nonrenewable resources: $\forall k \in K$, Q_k denotes the initial level of resource k. Each activity $i \in Y$ consumes b_{ik}^- units of resource k at its beginning, and produces b_{ik}^+ units of k at its end. If $b_{ik}^- \leq b_{ik}^+$, then activity i produces $(b_{ik}^+ - b_{ik}^-)$ units of resource k, whereas if $b_{ik}^- > b_{ik}^+$, then it consumes $|b_{ik}^+ - b_{ik}^-|$ units of resource k. The total amount of each nonrenewable resource must remain non-negative throughout the scheduling horizon.

A schedule $\mathcal{S}$ is a function assigning a starting time S_i to each activity $i \in Y$. It is feasible if it satisfies the precedence and resource constraints. Thus, solving RCPSP/CPR means computing a feasible schedule $\mathcal{S}$ with a minimal makespan.

1.5 Conclusion

In this chapter we have presented some definition and terminology dedicated to scheduling problems. Then we have presented some lower bounds, exact procedures and heuristics for solving the RCPSP. Moreover, we have studied three scheduling problems with nonrenewable resources. The first problem is the financing problem which can be solved using the shifting algorithm, the second problem is the Project Scheduling Problem with Inventory Constraints of [Neumann and Schwindt, 2002] and the last one is the RCPSP with consumption and production of resources.

Project Scheduling Problem with Inventory Constraints and RCPSP/CPR are two generalizations of RCPSP. In the next chapter, we introduce another extension of RCPSP called the Extended Resource Constrained Project Scheduling Problem (ERCPSP). ERCPSP is a general scheduling problem where the availability of resources is depleted and replenished. An instance of this problem consists of events,

nonrenewable resources and generalized precedence constraints between pairs of events.

The Extended RCPSP

Sommaire

2.1 Introduction **23**
2.2 Problem formulation **25**
2.3 Decision and Resource Usage Problems **29**
2.4 Linear orders for ERCPSP **29**
2.5 Polynomial cases of ERCPSP **32**
2.6 Dynamic Programming: parallel chain case **50**
2.7 Conclusion **51**

2.1 Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is the basic problem type in project scheduling. It is without doubt the most widely studied scheduling problem in literature, and it has resulted in an overwhelming amount of papers with solution procedures devoted to it. In this problem, non-preemptive activities requiring *renewable* resources, and subject to precedence constraints, have to be scheduled in order to minimize the makespan. The RCPSP with general time lag constraints has also been the subject of several papers [Bartusch et al., 1988] [Cesta et al., 2002] [Neumann and Zhan, 1997] [Neumann et al., 2006]. It also concerns only renewable resources such as the workforce. Renewable resources are allocated to activities at their starting times and released at their completion times. On the contrary, a *nonrenewable* resource is produced or consumed by an activity at its starting time only. Money is an example of nonrenewable resource for which Carlier and Rinnooy Kan introduced the financing problem [Carlier and Rinnooy Kan, 1982].

In this work we address the Extended Resource Constrained Project Scheduling Problem (ERCPSP). ERCPSP is a general scheduling problem where the availability

of resources is depleted and replenished [Carlier et al., 2009]. An instance of ERCPSP consists of events, nonrenewable resources and generalized precedence constraints between pairs of events. Each event produces or consumes some units of resources at its occurrence time. The objective is to build a schedule that satisfies the precedence and resource constraints and minimizes the makespan.

ERCPSP is a generalization of RCPSP where activities requiring renewable resources are replaced by events consuming or producing nonrenewable resources. Some other authors have worked on models similar to ERCPSP. We can quote the works of Neumann and Schwindt [Neumann and Schwindt, 2002] and of Laborie [Laborie, 2002]. Neumann and Schwindt formalized the Project Scheduling Problem with Inventory Constraints where the availability of each resource is at any time upper and lower bounded. To solve this problem, they proposed a branch-and-bound algorithm with a filtered beam search heuristic. Beck [Beck, 2002] proposed heuristics for constraint-directed scheduling with inventory which exploit dynamic constraint criticality. Laborie [Laborie, 2002] introduced the concept of a Resource Temporal Network (RTN). He proposed a constraint propagation algorithm to solve the problem. Bouly *et al.* [Bouly et al., 2005] developed a model which allows resource production by tasks, and provided algorithms to solve the problem for makespan minimization. Moreover, Sourd and Rogerie [Sourd, 2005] introduced continuous reservoir model, in which activities fill or empty the reservoir at a constant rate from their start time to their completion time. A branch-and-bound method for solving scheduling problems with continuous reservoirs can be found in Neumann et al. [Neumann et al., 2005]. Koné et al. [Koné et al., 2013] worked on the RCPSP with Consumption and Production of Resources (RCPSP/CPR). The particularity of their extension of RCPSP is that, in addition to renewable resources considered in the basic version, it also involves nonrenewable resources which can be consumed (or not) at the starting time of an activity in a certain amount and/or then produced in another amount at the completion time of this activity. To solve this problem, Koné et al. proposed four mixed integer linear programming models for RCPSP/CPR. ERCPSP coincides with the problem considered by [Neumann and Schwindt, 2002] and [Laborie, 2002] where no upper bound on the resource availability is prescribed.

Several methods have been introduced to enumerate solutions of RCPSP [Brucker et al., 1999] [Kolisch and Padman, 2001]. One of them is based on the notion of complete *linear order* of activities, which corresponds to the order of their execution. Linear order was termed *arbitrage* by Carlier in [Carlier, 1984] because for every linear order we can compute an earliest schedule which respects it or we can prove its infeasibility. In a recent paper Carlier et al. [Carlier et al., 2009] have

generalized this notion to the ERCPSP. They showed how to associate an earliest schedule with a linear order. The drawback of this approach is, of course, the large number of linear orders. So they considered only linear orders of consumption events. They show that there also exists an earliest schedule, and they described polynomial algorithms to compute it.

The remaining of this paper is structured as follows. In Section 2.2 we present some terminology dedicated to basic concepts and formulate the ERCPSP, and we show the connection between our problem and other scheduling problems with production and consumption of resources. In Section 2.3 we present the Decision and the Resource Usage Problems. In Section 2.4 we report an algorithm which computes the earliest schedule of a complete linear order, and we present the notion of linear order on consumption events. In Section 2.5 we present four special cases of ERCPSP for which the decision problem can be solved in polynomial time. In Section 2.6 we introduce a dynamic programming algorithm to solve the ERCPSP with parallal chain precedence graph, and finally we conclude this chapter in Section 2.7.

2.2 Problem formulation

An instance $I = (X, K, U, a, v)$ of the Extended Resource Constrained Project Scheduling Problem consists of:

- A set $X = \{0, 1, ..., n, n+1\}$ of events: $\forall i \in X$, the occurrence time of event i is denoted $S(i)$ (also denoted S_i). Of course $S(i)$ is not given and has to be determined. By convention, the two events 0 and $n+1$ are added to respectively define the start and the end of the schedule.

- A set U of precedence constraints which express relations of start-to-start between pairs of events. $\forall (i, j) \in U$, the precedence constraints have the form $S_i + v_{ij} \leq S_j$, where v_{ij} represents the time lag between events i and j. If $v_{ij} < 0$, this implies that event i has to occur no later than time $S_j - v_{ij}$, whereas if $v_{ij} \geq 0$, then event j cannot occur before time $S_i + v_{ij}$.

- A set K of non-renewable resources: $\forall k \in K$, the initial level of resource k is denoted by Q_k and $\forall i \in X, \forall k \in K$, a_i^k represents the quantity of resource k produced or consumed by event i. If a_i^k is positive, then event i produces the quantity a_i^k of resource k, whereas if $a_i^k \leq 0$, it consumes the quantity $|a_i^k|$ of resource k. We denote by $X_k^p = \{e \in X | a_e^k > 0\}$ (resp. $X_k^c = \{e \in X | a_e^k < 0\}$)

the set of production (resp. consumption) events of resource k. At any time, the resource availability must be positive or null for each resource $k \in K$.

We say that an event i is a *direct predecessor* of an event j if there exists a non-negative arc from i to j in the graph (X, U), which is equivalent to say that j is a *direct successor* of i. $l_{i,j}$ denotes the length of the longest path from i to j. We say that an event i is an *ascendant* of an event j if there exists a path from i to j with non-negative $l_{i,j}$, which is equivalent to say that j is a *descendant* of i. Event j is a *0-descendant* if $l_{i,j} = 0$ and *strict-descendant* if $_{i,j} > 0$. We denote the set of direct successors of an event i as $\Gamma^+(i)$, and the set of all descendants of i, not including i, as $\bar{\Gamma}^+(i)$. The corresponding sets of direct predecessors and ascendants are denoted respectively as $\Gamma^-(i)$ and $\bar{\Gamma}^-(i)$. Thus, event 0 (resp. event $n+1$) is an ascendant (resp. a descendant) of all the other events. Moreover, we set $S_0 = 0$, $a_0^k = Q_k$ and $a_{n+1}^k = 0$ for each $k \in K$.

A *schedule* S on event set X is a function assigning an occurrence time S_i to each event $i \in X$. The makespan of a schedule S can be computed as $C_{max} = S_{n+1}$. A schedule is feasible if it satisfies the precedence constraints

$$S_i + v_{ij} \leq S_j \qquad \forall (i,j) \in U$$

and the resource constraints

$$\sum_{i \in X(S,t)} a_i^k \geq 0 \qquad \forall k \in K, \forall t \in H$$

where $X(S,t) = \{i \in X \mid S_i \leq t\}$ is the set of events which have occurred by time $t \geq 0$, the set $H = \{0, 1, ..., T\}$ is the scheduling horizon, and T is some given upper bound on the makespan, which means that all events have to occur no latter than time T. An optimal schedule is a feasible schedule which minimizes the makespan. So, the Extended RCPSP can be formulated as follows:

$$\begin{aligned}
&\text{Minimize} && S_{n+1} && && (2.1)\\
&\text{subject to} && \sum_{i \in X(S,t)} a_i^k \geq 0 && \forall k \in K, \forall t \in H && (2.2)\\
& && S_i + v_{ij} \leq S_j && \forall (i,j) \in U && (2.3)\\
& && S_i \geq 0 && \forall i \in X \setminus \{0\} && (2.4)\\
& && S_0 = 0 && && (2.5)
\end{aligned}$$

The single-resource case of ERCPSP is defined by a quadruplet (X, U, a, v). In this case, a_i defines the number of resource units produced or consumed by event i,

a_0 corresponds to the initial resource units of the project, and the set of production (resp. consumption) events is denoted X^p (resp. X^c). In order to illustrate the presentation, we use the following example.

Exemple 2.1. *Let* $X = \{0, 1, 2, 3, 4, 5, 6\}$ *be the set of events.* $a_0 = +3, a_1 = +2, a_2 = -3, a_3 = -3, a_4 = +4, a_5 = -1, a_6 = 0, v_{13} = 3$ *(event* 1 *occurs at least 3 time units before event* 3*),* $v_{31} = -4$ *(event* 3 *cannot occur later than 4 times units after event* 1*),* $v_{24} = 6$*,* $v_{25} = 1$*,* $v_{01} = 0$*,* $v_{02} = 0$*,* $v_{36} = 0$*,* $v_{46} = 0$ *and* $v_{56} = 0$*.*

The graph resulting from Example 2.1 is shown in Fig 2.1. The number associated with a node represents the number of resource units required for that event, and the number corresponding to an arc represents the time lag. The number corresponding to event 0 is equal to the initial number of resource units for the project.

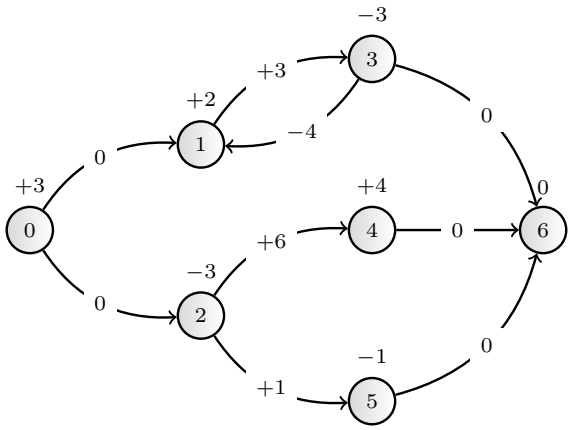


Figure 2.1 – An instance of ERCPSP with seven events and one resource

An optimal schedule for this instance is $S = \{S_0 = 0,\ S_1 = 2,\ S_2 = 0,\ S_3 = 6,\ S_4 = 6,\ S_5 = 2,\ S_6 = 6\}$. Fig 2.2 shows the resource availability over time associated with S.

2.2.1 Project Scheduling Problem with Inventory Constraints and ERCPSP

The ERCPSP can model a project scheduling problem with inventory constraints. Let us consider an instance of this problem and a resource $k \in K$. We replace resource k by two new resources denoted k_1 and k_2. If event i produces (resp. consumes) a quantity a_i^k in the original instance, it will produce (resp. consume) a_i^k units of k_1 in the new instance ($a_i^{k_1} = a_i^k$) and consume (resp. produce) a quantity a_i^k of resource k_2 ($a_i^{k_2} = -a_i^k$). To introduce the lower limit $\underline{Q_k}$ of resource k, we

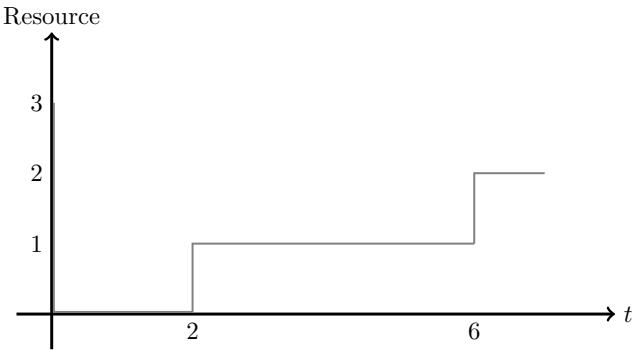


Figure 2.2 – Resource availability curve

add an event at the beginning of the project, which consumes $\underline{Q_k}$ units of resource k_1. To introduce the capacity $\overline{Q_k}$, we add an event at the beginning of the project which produces $\overline{Q_k}$ units of resource k_2.

2.2.2 RCPSP/CPR and ERCPSP

Two instances of ERCPSP I_1 and I_2 can be associated with each instance I of RCPSP/CPR. I_1 is obtained as follows. Each activity i is replaced by two events i' and i'', i' corresponds to the starting time of activity i and i'' to its completion time. To represent the processing time p_i, an arc (i', i'') valued by p_i and another arc (i'', i') valued by $-p_i$ are added. For each precedence relation $(i, j) \in E$, an arc (i'', j') valued by 0 is added. For each renewable resource $\rho \in R$, event i' consumes e_i^ρ units of resource ρ at its occurrence time ($a_{i'}^\rho = -e_i^\rho$), while event i'' produces e_i^ρ units of resource ρ ($a_{i''}^\rho = e_i^\rho$). To represent the global resource amount R_ρ, we add an event at the beginning of the project which produces R_ρ units of resource ρ. For each nonrenewable resource $k \in K$, event i' consumes b_{ik}^- units of k at its occurrence time ($a_{i'}^k = -b_{ik}^-$), while event i'' produces b_{ik}^+ units of k ($a_{i''}^k = b_{ik}^+$). To represent the initial level of resource k, we add an event at the beginning of the project which produces Q_k units of resource k. The instance I_2 is obtained using the same transformation except that no negative arc from i'' to i' is added.

Proposition 2.1. *Let I_1 and I_2 be two instances of ERCPSP associated with an instance I of RCPSP/CPR. I_1 and I_2 have the same optimal makespan which is also the optimal makespan of I.*

Proof. At first, there is a bijective correspondence between the schedules of I_1 and the schedules of I. Let S_1 be an optimal schedule for I_1 and S_2 an optimal schedule for I_2. The makespan of S_2 is a lower bound for I_1, since I_2 is a relaxation of I_1

obtained by removing all negative arcs. Thus, $S_2(n+1) \leq S_1(n+1)$.
A new schedule S'_2 can be obtained from S_2 by scheduling each consumption event i' at $S_2(i')$ and each production event i'' at $S_2(i') + p_i$, where i' and i'' are the two events associated with activity i. S'_2 respects all precedence and resource constraints of I_2 and I_1. Thus, it is an optimal schedule for I_2 ($S'_2(n+1) = S_2(n+1)$). It is also a feasible schedule for I_1 because i' is the only predecessor of i'' which is a production event and it is better to schedule a production event as soon as possible. So, $S'_2(n+1) \geq S_1(n+1)$. We conclude that I_1 and I_2 have the same optimal makespan. □

2.3 Decision and Resource Usage Problems

Let $I = (X, U, v, a)$ be an instance of ERCPSP with a single resource. The Decision Problem is determining whether I has a feasible schedule or not. The Resource Usage Problem consists of determining the smallest number of initial resources necessary for I to be feasible. The decision problem is NP-complete, inasmuch as determining the existence of feasible schedule for RCPSP with minimum and maximum time lags is NP-complete [Johannes, 2005]. The Decision Problem cannot be solved in polynomial time even if only positive arcs are permitted. In fact, the special case where for an instance $a_i \in \{-1, +1\}$, $1 \leq i \leq n$, and $v_{ij} = 1$, $\forall (i,j) \in U$, is equivalent to the cumulative cost problem. [Sethi, 1975] proved that such a problem is NP-complete.

However, in the case of some specific precedence constraints, the Decision Problem can be solved in polynomial time. The strategy is essentially to schedule the events which increase the level of resources as early as possible. In section 2.5, we present four cases in which the decision problem can be solved in polynomial time: the relocation problem, the parallel chain case, the series-parallel case and the interval order case.

2.4 Linear orders for ERCPSP

A linear order of the resource k is a set of conjunctive arcs $\alpha_k = \{(i_1, i_2), (i_2, i_3), ..., (i_{n-1}, i_n)\}$ with null valuations, while $\Pi_k = (i_1, i_2, ..., i_n)$ is a permutation on the set of events requiring the resource k [Carlier et al., 2009].

Definition 2.1. *A complete linear order is a linear order which contains all events of the project*

In order to simplify the presentation, we suppose that there is only one resource and that all events are in the linear order $\alpha = \{(1,2),(2,3),...,(n-1,n)\}$. The case of several resources is discussed in section 2.4.3. Moreover, when there is an arc from i to j with valuation ψ in the graph and we wish to add an arc from i to j with valuation φ in the graph, we will take the larger valuation among ψ and φ as the arc weight.

2.4.1 Compatibility

We say a complete linear order $\alpha = \{(1,2),(2,3),...,(n-1,n)\}$ is compatible if there exists a feasible schedule $S = \{S_1, S_2, ..., S_n\}$, such that for each arc (i,j) in α we have $S_j \geq S_i$. In order to test the compatibility of a complete linear order, we have to check the existence of a feasible schedule which satisfies resource constraints in the graph $G = (X, U \cup \alpha)$. [Carlier et al., 2009] changes this problem into a precedence constrained problem by introducing a set β of implicit precedence constraints as follows.

The resource constraint in our problem is that we can start an event only when there are sufficient resources for it. Now, let us assume that we wish to start event r and

$$\sum_{j=0}^{r} a_j < 0.$$

In this case, if there is no other event which produces resources occurring at the same time, the resource constraint will not be satisfied. So we have to force a production event t to occur at the same time as event r, where

$$\sum_{j=0}^{t} a_j \geq 0 \text{ and } t > r.$$

For $s,t \in \{1,...,n\}$ where $s+1 < t$, we say that the arc from $(t, s+1)$ valued by 0 is implied by α if and only if for every r where $s < r < t$ the following conditions are satisfied:

(1) $\sum_{j=0}^{s} a_j \geq 0$

(2) $\forall r \in]s,t[\ \sum_{j=0}^{r} a_j < 0$

(3) $\sum_{j=0}^{t} a_j \geq 0$.

β is the set of all arcs implied by α in this way. In order to generate β, let $B = \{j | \sum_{k=0}^{j} a_k \geq 0\}$. Without loss of generality, we suppose that $B = \{j_1, j_2, ..., j_m\}$ such that $j_1 < j_2 < ... < j_m$. If $j_1 \neq 1$, we add the arc $(j_1, 1)$ of zero valuation to β, and for every s such that $j_{s+1} > j_s + 1$, we add the arc $(j_{s+1}, j_s + 1)$ of zero valuation to β.

Proposition 2.2. *[Carlier et al., 2009] All schedules which respect precedence constraints of $U \cup \alpha \cup \beta$ are feasible schedules where β contains all implicit precedence constraints deduced from α, and conversely.*

In addition, if we modify the three conditions above (1), (2) and (3) as follows:

(1') $\sum_{j=0}^{s} a_j \leq \bar{Q}$,

(2') $\forall r \in]s,t[\sum_{j=0}^{r} a_j > \bar{Q}$,

(3') $\sum_{j=0}^{t} a_j \leq \bar{Q}$,

we can also have an upper bound $\bar{Q}$ on the level of resources. So this method can be generalized to the project scheduling problem with inventory constraints [Neumann and Schwindt, 2002] by adding arcs with zero weight to β.

2.4.2 EST schedule of a complete linear order

Algorithm 2.1, which takes a complete linear order as input, will yield the earliest start time schedule if $G(\alpha, \beta) = (X, U \cup \alpha \cup \beta)$ has no directed cycle of positive length [Carlier et al., 2009]. The Earliest Starting Time (EST) schedule can be computed using the Modified Label Correcting Algorithm [Ahuja et al., 1995].

Algorithm 2.1: Computation of the earliest start time schedule for a complete linear order.

Input:
A complete linear order $\alpha = \{(i_1, i_2), (i_2, i_3), ..., (i_{n-1}, i_n)\}$.
An instance $I = (X, U, a, v)$.
Output: An Earliest Start Time Schedule S for the linear order α

Generate the set β of all implicit precedence constraints deduced from α;
Use the Modified Label Correcting Algorithm to compute S in graph $G(\alpha, \beta)$;
return S

Theorem 2.1. *[Carlier et al., 2009] Algorithm 2.1 returns an optimal schedule in $O(n \times (|U| + |\alpha| + |\beta|))$ if the directed graph $G(\alpha, \beta) = (X, U \cup \alpha \cup \beta)$ contains no directed cycle of strictly positive length.*

2.4.3 Case of multiple resources

The case of multiple resources is more complicated than the one resource problem. For two resources the feasibility problem has been shown to be NP-complete [Neumann et al., 2003]. However, the algorithm with complete linear order for one

resource can be extended to multiple resources directly. We only have to generate implicit precedence constraints set β_k for each resource k separately and then compute the EST schedule in the graph $(X, U \cup \alpha \cup \beta_1 \cup \beta_2 ... \cup \beta_R)$, where R is the number of resources.

2.4.4 Linear order of consumption events

A consumption linear order $\alpha^c = \{(i_1, i_2), (i_2, i_3), ..., (i_{c-1}, i_c)\}$ is the linear order of all consumption events. We denote $G = (X, U \cup \alpha^c)$ as $G(\alpha^c)$.

Theorem 2.2. *[Carlier et al., 2009] For a given consumption linear order, there exists an earliest start time schedule provided that there exists at least one feasible schedule.*

A production linear order $\alpha^p = \{(i_1, i_2), (i_2, i_3), ..., (i_{p-1}, i_p)\}$ is the linear order of all production events. Theorem 2.2 is replaced by Theorem 2.3

Theorem 2.3. *[Carlier et al., 2009] For a given production linear order, there exists a latest start time schedule provided that there exists at least one feasible schedule.*

2.5 Polynomial cases of ERCPSP

In this section, we present four cases of ERCPSP for which the decision problem can be solved in polynomial time: the relocation problem, the parallel chain case, the series-parallel case and the interval order case.

2.5.1 The Relocation Problem

The relocation problem consists of a set Y of n activities, which have to be scheduled on one machine without preemption. The objective is the minimization the makespan [Kaplan and Amir, 1988, Lin and Cheng, 1999]. Each activity $i \in Y$ of duration p_i acquires a quantity a_i^- of resources at its starting time S_i, and returns a quantity a_i^+ of resources at its completion time C_i ($C_i = S_i + p_i$). In general, a_i^+ can be less than, equal to, or greater than a_i^-. All activities are assumed to require the same type of resource, the initial number of available resources is denoted V_0. A schedule S is a function assigning a starting time S_i to each activity $i \in Y$. The makespan of a schedule S can be computed as $C_{max} = max_{i \in Y}(S_i + p_i)$. A schedule is feasible if each activity following the schedule can be successfully processed. An instance of the relocation problem is feasible if it admits some feasible schedule. Note that, for fixed

values of a_i^- and a_i^+ such that $\sum_{i \in Y} a_i^- \geq \sum_{i \in Y} a_i^+$, the feasibility of a relocation problem is determined by V_0.

Each instance of this problem can be associated with an instance of ERCPSP. Each activity $i \in Y$ is represented by two events i and i' with $a_i^0 = -a_i^-$, $a_i^1 = -1$, $a_{i'}^0 = a_i^+$, $a_{i'}^1 = +1$, and two arcs $v_{ii'} = p_i$ and $v_{i'i} = -p_i$; however, the arc $v_{i'i}$ can be ignored. So an instance of the relocation problem $RP = (Y, V_0, a^-, a^+, p)$ where $Y = \{1, 2, ..., n\}$ can be represented by an instance $I = (X, U, K, a, v)$ of ERCPSP with two resources and no negative time lags, where $X = \{0, 1, 1', 2, 2', ..., n, n', n+1\}$, $Q_0 = a_0^0 = V_0$ and $Q_1 = 1$. Note that the resource 1 here insures the sequential execution of the events associated with each activity. In the conjunctive graph $G = (X, U)$, all consumption events have only one direct predecessor corresponding to the fictitious beginning event 0, and all production events have only one direct successor corresponding to the fictitious termination event $n+1$. Each consumption event i has only one direct successor i' and each production event i' has only one direct predecessor i. Note that the relocation problem can be considered as a problem with parallel chain precedence constraints where there are only two events in each chain.

2.5.1.1 Feasibility test of Kaplan and Amir

Kaplan and Amir introduced a simple method for determining the feasibility of the relocation problem. For given values of a_i^- and a_i^+, their idea is to determine the smallest number of resources V^* such that a relocation problem with the given values of a_i^- and a_i^+ remains feasible. The original problem is feasible if and only if $V_0 \geq V^*$ [Kaplan and Amir, 1988]. To determine V^*, they proposed a method to construct a particular non-overlapping sequential schedule S^{RP} with the following property: If a relocation problem with V_0 initial resources (given the values of a_i^- and a_i^+) admits some feasible schedule, then S^{RP} is also feasible for that problem. The minimum number of resources necessary for this particular schedule to be feasible is equal to V^*.

Let Y^+ and Y^- be two subsets of Y, such that $i \in Y^+$ if and only if $a_i^- \leq a_i^+$, and $i \in Y^-$ if and only if $a_i^- > a_i^+$. The schedule S^{RP} is constructed by sequencing all activities $i \in Y^+$ in nondecreasing order of their a_i^- followed by all activities $i \in Y^-$ in nonincreasing order of their a_i^+. More precisely:

- For $i, j \in Y^+$, i precedes j in S^{RP} iff $a_i^- \leq a_j^-$.
- For $i, j \in Y^-$, i precedes j in S^{RP} iff $a_i^- \geq a_j^-$.
- For $i \in Y^+$, $j \in Y^-$, i precedes j in S^{RP}.

Let s_i be the ith activity scheduled in S^{RP}. The schedule S^{RP} is feasible for the relocation problem with V_0 initial resources, if and only if the following constraints are satisfied:

$$\sum_{j=1}^{i} a_{s_j}^{-} - \sum_{j=1}^{i-1} a_{s_j}^{+} \leq V_0, \quad i = 1, 2, ..., n \tag{2.6}$$

From (2.6) it is easy to see that the minimum number of resources V^* necessary to maintain the feasibility is given by:

$$V^* = \max_{1 \leq i \leq n} \{ \sum_{j=1}^{i} a_{s_j}^{-} - \sum_{j=1}^{i-1} a_{s_j}^{+} \} \tag{2.7}$$

Thus, to test the feasibility of a relocation problem, we construct the schedule S^{RP}. Then we compute V^* from (2.7). If $V_0 \geq V^*$, the problem is feasible.

2.5.1.2 Johnson's rule

Kaplan and Amir [Kaplan and Amir, 1988] show that the relocation feasibility problem is equivalent to the two-machine flowshop problem [Johnson, 1954] which can be solved in $O(n \log n)$. The two-machine flowshop problem consists of scheduling n activities on two machines with the objective of minimizing makespan. The processing time of activity i, $i = 1, 2, ..., n$ on machine j, $j = 1, 2$, is denoted p_{ij}. Johnson proved that the optimal schedule is given by the following algorithm:

Algorithm 2.2: Algorithm of Johnson

(1) Let $Y^1 = \{i | p_{i1} \leq p_{i2}\}$ and $Y^2 = \{i | p_{i1} > p_{i2}\}$;
(2) Arrange the activities in set Y^1 in nondecreasing order of p_{i1};
(3) Arrange the activities in set Y^2 in nonincreasing order of p_{i2};
(4) Construct an optimal sequence: the ordered set Y^1 followed by the ordered set Y^2;

To see the connection between the relocation feasibility problem and the problem considered by Johnson, we suppose that it takes one unit of time to produce (resp. consume) any resource, and that resources are produced (resp. consumed) sequentially. Therefore, each activity in the relocation problem can be considered as an activity which needs to be processed on two machines. The processing time on the first machine corresponds to its resource consumption, thus $p_{i1} = a_i^-$, while the processing time on the second machine corresponds to its resource production, thus $p_{i2} = a_i^+$. The makespan of the optimal schedule in the problem of Johnson is given

by:

$$C_{max} = \sum_{i=1}^{n} p_{i2} + Idle \tag{2.8}$$

where $Idle$ is the idle time on the second machine. From (2.8), the way to minimize the makespan in Johnson's problem is to minimize $Idle$. Now, suppose that one unit of time is necessary to construct each of the V_0 resources. Thus in the relocation context (2.8) is equivalent to:

$$C_{max} = \sum_{i=1}^{n} a_i^+ + V_0 \tag{2.9}$$

As we can see, the makespan in Johnson's problem is identical to the total number of resources produced in the relocation problem (new resources plus V_0). This total is minimized when V_0 is minimized, which is precisely the relocation feasibility problem.

2.5.2 The parallel chain case

In this section we investigate a special case of ERCPSP with single resource, where the precedence graph $G = (X, U)$ consists of a set of SP ($\geq$ 2) parallel chains. This special case is an extension of the problem considered by [Abdel-wahab and Kameda, 1978], where more than one event can be executed at the same time. Abdel-wahab and Kameda introduced an algorithm for minimizing maximum cumulative cost. This algorithm calculates the change of resource level, then determines production and consumption subchains in each chain. An optimal schedule is obtained by merging the production subchains in nondecreasing order of their rises followed by consumption subchains in nonincreasing order of their drops. Thus, a dominant chain which minimizes the maximum cumulative cost is obtained.

In this section, we adapt this algorithm to our problem. It always consists of determining the minimum required amount of initial resources. Abdel Wahab and Kameda sequenced the events because they are executed on one machine. In our problem there is no machine. Thus, some events can be executed at the same time. The algorithm is actually very similar. It is also based on a decomposition of chains into production and consumption subchains. We will see that these subchains can be seen as jobs of a flow-shop with two machines. The idea is to construct a schedule of standard form, where the events of each subchain are scheduled next to each other, from any feasible schedule. Then, we can apply the Johnson's rule to these subchains in order to obtain an optimal sequence. This method will be illustrated

by the example of Fig 2.3.

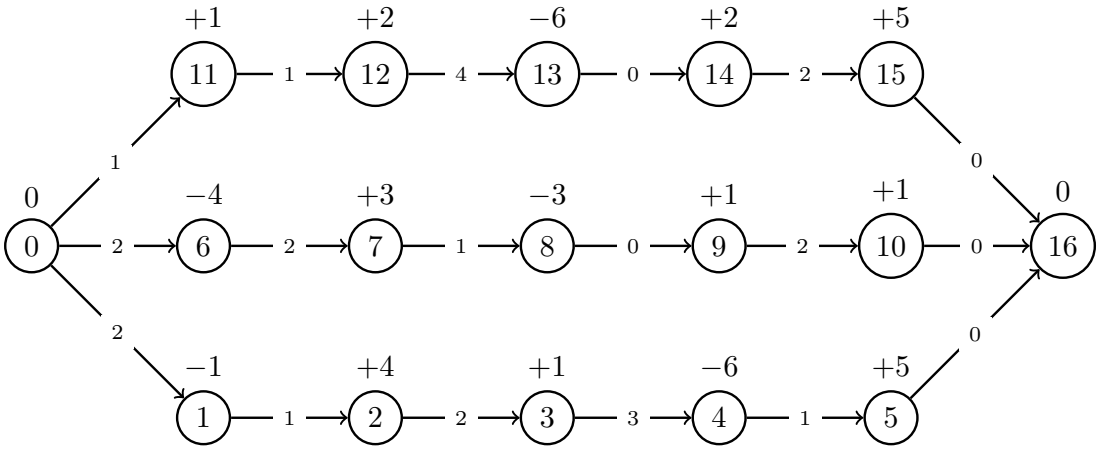


Figure 2.3 – ERCPSP instance with three chains

2.5.2.1 Notation

The notation that follows is used for this special case. Suppose that chain h ($1 \leq h \leq SP$) contains L_h (≥ 1) events, $C_1^h, C_2^h, ..., C_{L_h}^h$, in the order of precedence constraints. We use $b_h(i)$ to denote the quantity of resources produced or consumed by event C_i^h. For each chain h, the events C_0^h and $C_{L_h+1}^h$ correspond respectively to the fictitious events 0 and $n+1$. The earliest occurrence time of C_i^h is denoted $ES_h(i)$ ($ES_h(i) = ES(C_i^h)$). For each chain h ($1 \leq h \leq SP$) we introduce the resource level function $A_h : \{0, 1, ..., L_h\} \longrightarrow \mathbf{Z}$, defined by:

$$A_h(i) = \sum_{\{j | ES_h(j) \leq ES_h(i)\}} b_h(j) \quad \text{for} \quad 1 \leq i \leq L_h \quad \text{and} \quad A_h(0) = 0. \tag{2.10}$$

$A_h(i)$ is the level of resource at the earliest occurrence time of event C_i^h. Fig 2.3 gives an instance of ERCPSP consisting of three chains (i.e., $SP = 3$) and a total of seventeen events. Fig 2.4 shows the three parallel chains associated with this instance. Table 2.1 illustrates the values of the resource level functions for the example of Fig 2.3.

Let S be a schedule. The level of resource at the occurrence time of event $e \in X$ is given by $A(e|S)$.

$$A(e|S) = \sum_{\{e' \ | \ S(e') \leq S(e)\}} a_{e'} \quad \text{for} \quad 1 \leq e \leq n \quad \text{and} \quad A(0|S) = a_0. \tag{2.11}$$

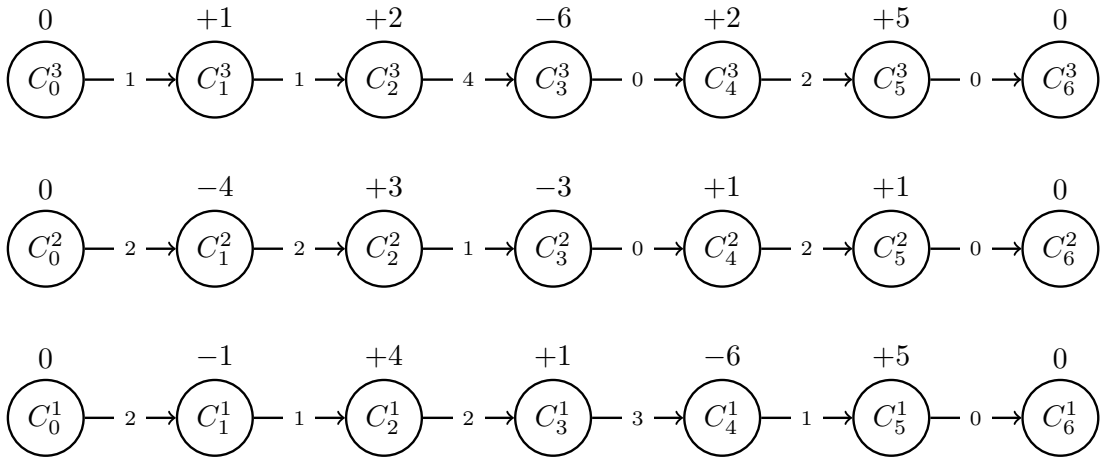


Figure 2.4 – Three parallel chains

i	0	1	2	3	4	5
$A_1(i)$	0	-1	+3	+4	-2	+3
$A_2(i)$	0	-4	-1	-3	-3	-2
$A_3(i)$	0	+1	+3	-1	-1	+4

Tableau 2.1 – The resource level functions of the example of Fig 2.3

The minimal level of resource $m(S)$ associated with S is given by:

$$m(S) = \min\{A(e \mid S) \quad | \quad e \in X\}. \tag{2.12}$$

Let I be an instance of ERCPSP with single type of resources. The resource usage problem consists of finding a schedule S that maximizes $m(S)$ and respects all precedence constraints. If $Q_0 \geq -m(S)$ then S is a feasible schedule for I.

Without loss of generality, we suppose that in each chain, if any event is followed by a consumption event, then the time lag between these two events is strictly positive. In the case we have a zero time lag between an event i and a consumption event c, we change the value of v_{ic} to 1 (c is executed strictly after i). This will not change the solution of the decision and the resource usage problems.

2.5.2.2 Decomposition of chains into optimal subchains

The decomposition of a chain provides a subsequence of production subchains, followed by a subsequence of consumption subchains. One of the two subsequences

may be empty. For example, the first chain of the instance of Fig 2.3 consists of one production subchain and one consumption subchain. The second chain consists of two consumption subchains, and the third chain consists of two production subchains (see Fig 2.5).

Each subchain starts by a resource consumption (fall) and finishes by a production (rise). For example, the subchain OP_1 needs at the beginning one unit of resource in order to produce 5 units of resource at the end. Therefore, its overall production is positive and it is equal to 4. This is why we talk about production subchain. Here, the event C_3^1 plays a special role because it corresponds to the maximal level of resource. The event C_1^1 also plays a special role (subchain pivot), because it corresponds to the minimal level of resource. Each subchain can be seen as an activity which needs to be processed on two machines. The processing time on the first machine corresponds to its fall, while the processing time on the second machine corresponds to its rise. Algorithmically, we first build the production subchains of a chain, then the consumption subchains.

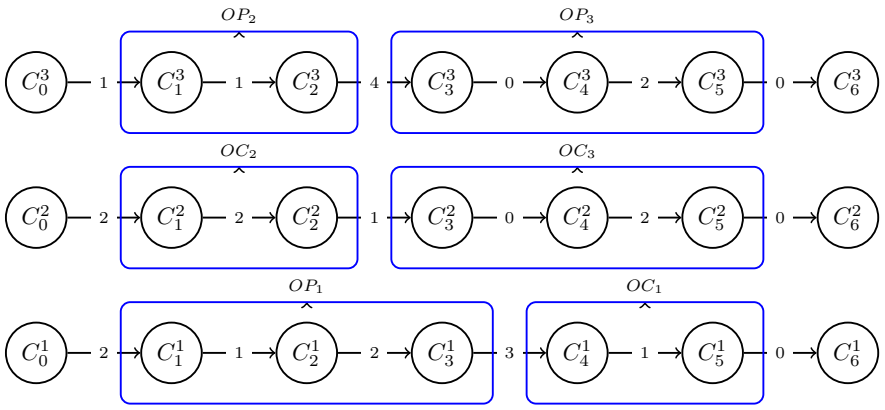


Figure 2.5 – OP-, and OC-subchains of the example of Fig 2.3

Let us consider a subchain α of chain h consisting of events $C_{u+1}^h, ..., C_p^h, ..., C_v^h$ $(0 \leq u < p \leq v \leq L_h)$ such that:

$$\begin{cases} A_h(u) \geq A_h(i) \geq A_h(p), & u \leq i \leq p. \\ A_h(p) \leq A_h(i) \leq A_h(v), & p \leq i \leq v. \end{cases} \tag{2.13}$$

The index p is called the pivot index. It corresponds to the index of event that minimizes $A_h(i)$ within the subchain α. The fall Δ_α^- of α is defined by $\Delta_\alpha^- = A_h(u) - A_h(p)$. The rise Δ_α^+ of α is defined by $\Delta_\alpha^+ = A_h(v) - A_h(p)$. If $\Delta_\alpha^+ \geq \Delta_\alpha^-$ then α is a production subchain (P-subchain), whereas if $\Delta_\alpha^+ < \Delta_\alpha^-$ then α is a consumption subchain (C-subchain). The subchains of a chain which respect the Johnson's rule

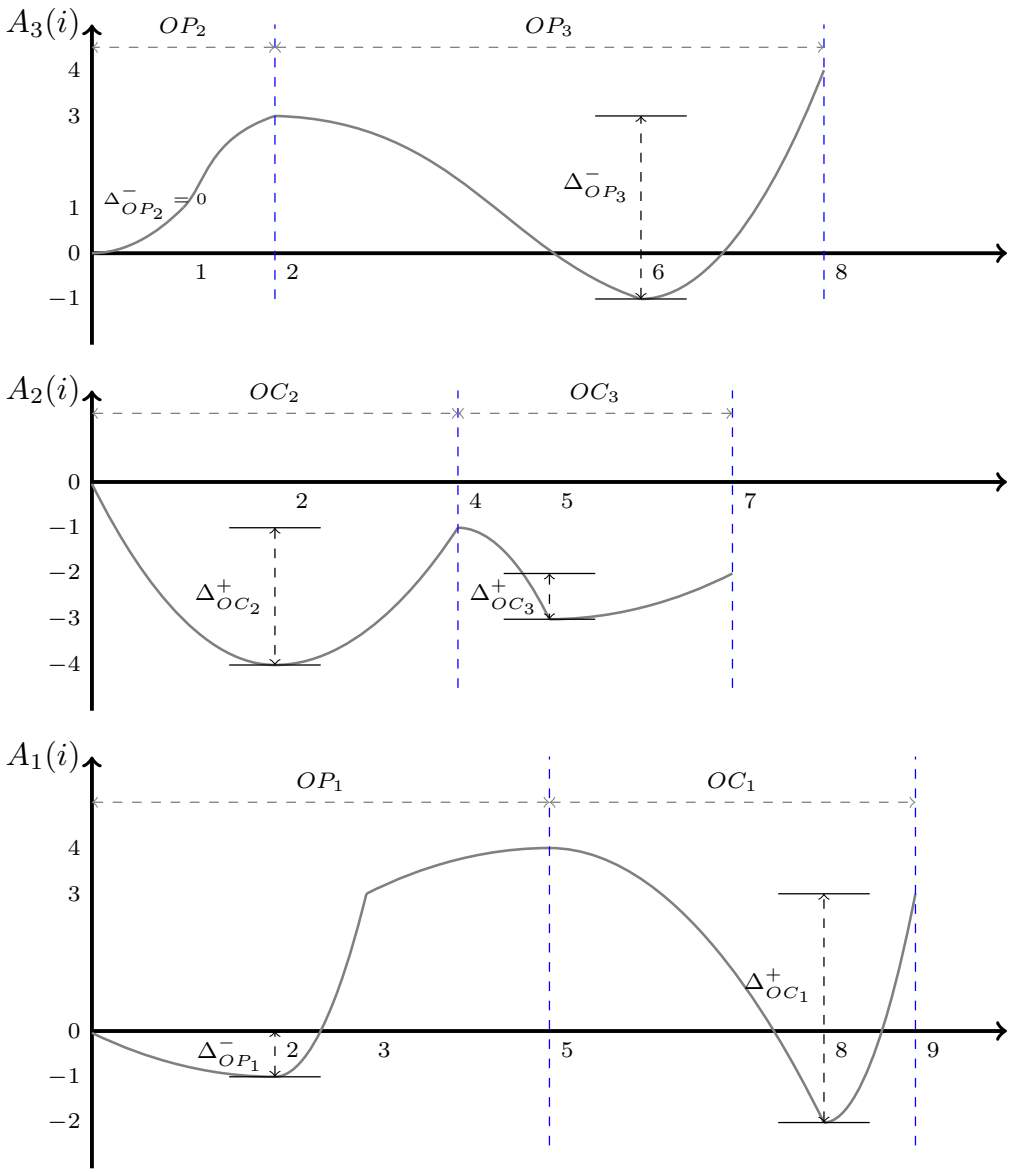


Figure 2.6 – The resource level functions

are called optimal subchains (The fall (resp. the rise) monotonically decreases (resp. increases) for the successive P-subchains (resp. C-subchains) of each chain). An OP-subchain is an optimal production subchain and an OC-subchain is an optimal consumption subchain. Algorithm 2.3 (resp. Algorithm 2.4) determines the OP-subchains and (resp. OC-subchains) of each chain.

Fig 2.5 shows the optimal production and consumption subchains associated with the instance of Fig 2.3. As we can see, any chain consists of zero or more OP-subchains, followed by zero or more OC-subchains. The fall (resp. the rise) monotonically decreases (resp. increases) for the successive OP-subchains (resp. OC-

Algorithm 2.3: Construction of OP-subchains

1) Determine the event which provides the maximal production;
   $v \leftarrow \max\{0 \leq i \leq L_h \mid A_h(i) = \max\{A_h(j) | 0 \leq j \leq L_h\}\}$;
   If $v = 0$, then EXIT: there is 0 OP-subchain;

2) Determine the event which minimizes the level of resource before C_v^h;
   $p \leftarrow \min\{0 < i \leq v \mid A_h(i) = \min\{A_h(j) | 0 < j \leq v\}\}$;

3) Determine the event which provides the maximal production before C_p^h;
   $w \leftarrow \min\{0 \leq i \leq p \mid A_h(i) = \max\{A_h(j) | 0 \leq j \leq p\}\}$;
   $u \leftarrow \max\{w \leq i \leq p \mid ES_h(i) = ES_h(w)\}$;

4) $C_{u+1}^h, C_{u+2}^h, ..., C_v^h$ form an OP-subchain;
5) If $u > 0$, then $v \leftarrow u$ and go to step (2);

Algorithm 2.4: construction of OC-subchains

1) Determine the event which provides the maximal production;
   $u \leftarrow \max\{0 \leq i \leq L_h \mid A_h(i) = \max\{A_h(j) | 0 \leq j \leq L_h\} + 1\}$;
   If $u \geq L_h$, then EXIT: there is 0 OC-subchain;

2) Determine the event which minimizes the level of resource after C_u^h;
   $p \leftarrow \max\{u \leq i \leq L_h \mid A_h(i) = \min\{A_h(j) | u \leq j \leq L_h\}\}$;

3) Determine the event which provides the maximal production after C_p^h;
   $v \leftarrow \max\{p \leq i \leq L_h \mid A_h(i) = \max\{A_h(j) | p \leq j \leq L_h\}\}$;

4) $C_u^h, C_{u+1}^h, ..., C_v^h$ form an OC-subchain;
5) If $v < L_h$, then $u \leftarrow v + 1$ and go to step (2);

subchains) of each chain (see Fig 2.6).

2.5.2.3 Standard Form Schedule

A schedule of standard form is a schedule in which all events of each subchain are scheduled next to each other. It can be obtained from any schedule by clustering the events of each optimal subchain around the event with the pivot index as follows:

- All the events not belonging to the subchain, which are scheduled before the pivot event, does not change.

- All the events not belonging to the subchain, which are scheduled strictly after the pivot event, are shifted by $2 * \Delta$, where Δ is equal to the length of the subchain.

- The pivot index is shifted by Δ.

- All the events belonging to the subchain, which are before the pivot event, are scheduled (as late as possible) immediately before the pivot event.

- All the events belonging to the subchain, which are after the pivot event, are scheduled (as soon as possible) immediately after the pivot event.

More formally, let $C^h_{u+1}, C^h_{u+2}, ..., C^h_p, ..., C^h_v$ be the subchain of chain h under consideration, where p is its pivot index. Let S and S' be two schedule, such that S' is obtained from S by clustering the events of the subchain around the pivot event. S' is defined as follows:

$$S'(e) = \begin{cases} S(e) + 2 * \Delta & \text{if } e \notin \{C^h_{u+1}, ..., C^h_v\} \text{ and } S(e) > S(C^h_p) \\ S(C^h_p) + \Delta + ES(e) - ES(C^h_p) & \text{if } e \in \{C^h_{u+1}, C^h_{u+2}, ..., C^h_v\} \\ S(e) & \text{otherwise.} \end{cases} \tag{2.14}$$

Theorem 2.4. *Let S be a schedule for an instance I of ERCPSP with parallel chains. Let S' be a schedule obtained from S by clustering the events of any OP-subchain or OC-subchain around the event with the pivot index. Then, we have $m(S') \geq m(S)$.*

Proof. Let $C^h_u, C^h_{u+1}, ..., C^h_p, ..., C^h_v$ the subchain of chain h under consideration, where p is its pivot index. S' is obtained from S by scheduling the events $C^h_u, C^h_{p+1}, ..., C^h_v$ around event C^h_p. If S respects all the precedence constraints then also S' respects them.

Let e be an event of X such that $S(e) < S(C^h_u)$ or $S(e) > S(C^h_v)$. By definition of S', we have $A(e \mid S') = A(e \mid S)$. In fact, all the events which are executed before or at time $S(e)$ by S, are also executed before or at time $S'(e)$ by S'. From this it follows that

$$A(e|S') \geq m(S), \quad e \in X \text{ and } S(e) \in [0, S(C^h_u)[\cup]S(C^h_v), S(n+1)]. \tag{2.15}$$

In the time interval $[S'(C^h_{u+1}), S'(C^h_v)]$ only events of chain h are scheduled by S'. Thus, we deduce from this and 2.13 that:

$$A(C^h_p|S) = A(C^h_p|S') \leq A(C^h_i|S') \leq A(C^h_v|S'), \quad p \leq i \leq v. \tag{2.16}$$

$$A(C^h_{u+1}|S') \geq A(C^h_i|S') \geq A(C^h_p|S') = A(C^h_p|S), \quad u+1 \leq i \leq p. \tag{2.17}$$

From this it follows that

$$A(C_i^h|S') \geq m(S), \quad u+1 \leq i \leq v. \tag{2.18}$$

Let e be an event not belonging to chain h such that $S(C_p^h) < S(e) \leq S(C_v^h)$ ($S'(e) > S'(C_v^h)$). Let $C_w^h, C_{w+1}^h, ..., C_v^h$, where $w \geq p+1$, be the events of h which are scheduled after e by S and before e by S'. So we have $A(e \mid S') = A(e \mid S) + \sum_{i=w}^{v} b_h(i)$. Now let us prove that $\sum_{i=w}^{v} b_h(i) \geq 0$.
If $ES(C_{w-1}^h) < ES(C_w^h)$ then $\sum_{i=w}^{v} b_h(i) = A_h(v) - A_h(w-1) \geq 0$.
If $ES(C_{w-1}^h) = ES(C_w^h)$ then $b_h(w) \geq 0$. Let w' be an index of chain h such that $ES(C_w^h) = ES(C_{w'}^h)$ and $ES(C_w^h) < ES(C_{w'+1}^h)$. The events belonging to $\{C_w^h, C_{w+1}^h, ..., C_{w'}^h\}$ are production events (a production event followed by a consumption event cannot have the same earliest occurrence time). This implies $\sum_{i=w}^{v} b_h(i) = \sum_{i=w}^{w'} b_h(i) + A_h(v) - A_h(w') \geq 0$. Therefore, we deduce that

$$A(e \mid S') \geq A(e \mid S) \geq m(S), e' \notin \{C_p^h, ..., C_v^h\} \text{ and } S(C_p^h) < S(e') \leq S(C_v^h). \tag{2.19}$$

Let e' be an event not belonging to chain h such that $S(e') \geq S(C_{u+1}^h)$ and $S'(e') < S'(C_{u+1}^h)$. Let $C_{u+1}^h, C_{u+2}^h, ..., C_w^h$, where $w \leq p$, be the events of h which are scheduled before e' in S and after e' in S'. So we have $A(e' \mid S') = A(e' \mid S) - \sum_{i=u+1}^{w} b_h(i)$. Now let us prove that $\sum_{i=u+1}^{w} b_h(i) \leq 0$.
If $ES_h(w) < ES_h(w+1)$ then $\sum_{i=u+1}^{w} b_h(i) = A_h(u) - A_h(w) \leq 0$.
If $ES_h(w) = ES_h(w+1)$ then $b_h(w+1) \geq 0$. Let w' be an index of chain h such that $ES_h(w) = ES_h(w')$ and $ES_h(w) < ES_h(w'+1)$. The events belonging to $\{C_{w+1}^h, C_{w+2}^h, ..., C_{w'}^h\}$ are production events (a consumption event followed by another consumption event cannot have the same earliest occurrence time). This implies $\sum_{i=u+1}^{w} b_h(i) = A_h(w') - A_h(u) - \sum_{i=w+1}^{w'} b_h(i) \leq 0$. Therefore, we deduce that

$$A(e' \mid S') \geq m(S), \ e' \notin \{C_{u+1}^h, ..., C_p^h\} \text{ and } S(C_{u+1}^h) \leq S(e') \leq S(C_p^h). \tag{2.20}$$

From (2.15), (2.16), (2.19) and (2.20) it follows that $m(S') \geq m(S)$. □

Theorem 2.4 implies that at least one optimal schedule is of standard form. Thus, we consider from now only the schedules of standard form.

2.5.2.4 Algorithm of resolution

The following algorithm constructs an optimal schedule for the Resource Usage Problem of this special case. The idea of this algorithm is to construct a schedule of

Algorithm 2.5: Algorithm to construction an optimal schedule

(1) Determine the OP-subchains and the OC-subchains of each chain;
(2) Schedule the OP-subchains sequentially in nondecreasing order of their rises;
(3) Schedule the OC-subchains sequentially in nonincreasing order of their falls;

standard form, where the events of each subchain are scheduled next to each other, from any feasible schedule. Then, we apply the Johnson's rule to these subchains in order to obtain an optimal sequence. This is stated in Theorem 2.5 resulting from Proposition 2.3. By applying Algorithm 2.5 to the example of Fig 2.3, we obtain an optimal schedule where the events of optimal subchains are sequenced in this order: $OP_2, OP_1, OP_3, OC_2, OC_3, OC_1$. Fig 2.7 represents the obtained optimal chain.

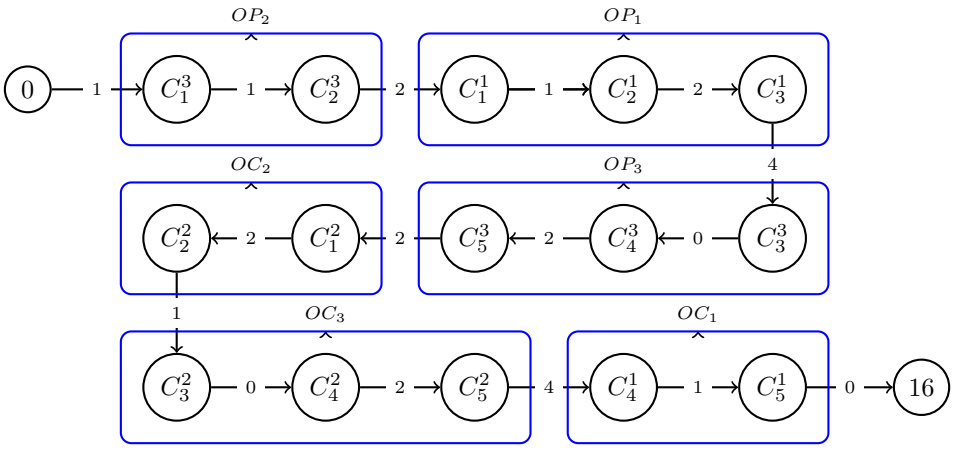


Figure 2.7 – Optimal chain

Theorem 2.5. *Let I be an instance of ERCPSP with parallel chains. An optimal schedule for the resource usage problem of I is obtained as follows. First we sequence the OP-subchains (if any) of each chain in nondecreasing order of their falls. Then we sequence the OC-subchains of each chain in nonincreasing order of their rises.*

In order to prove this theorem we need a few preliminary results. Let us introduce the rank $r(\alpha)$ of an optimal subchain α as follows.

$$r(\alpha) = \begin{cases} \frac{1}{\Delta_\alpha^- + 1}, & \text{if } \alpha \text{ is an OP-subchain} \\ \frac{-1}{\Delta_\alpha^+ + 1}, & \text{if } \alpha \text{ is an OC-subchain.} \end{cases}$$

Proposition 2.3. *Let S be a schedule of standard form, where an optimal subchain α of a chain is immediately followed by an optimal subchain β of another chain and $r(\alpha) \leq r(\beta)$. Let S' be schedule obtained form S by interchanging the position of α and β. Then we have $m(S') \geq m(S)$.*

Proof. S and S' have respectively the forms $SC_1.\alpha.\beta.SC_2$ and $SC_1.\beta.\alpha.SC_2$, where SC_1 and SC_2 are sequences of other optimal subchains. Suppose that α is consisting of events $C_u^h, ..., C_v^h$ and β is consisting of events $C_{u'}^{h'}, ..., C_{v'}^{h'}$. For each event $i \in X$ such that $S(i) \leq S(C_u^h)$ we have $S(i) = S'(i)$.

Let e be an event of X such that $S(e) \in [0, S(C_u^h)[\cup]S(C_{v'}^{h'}), S(n+1)]$. All the events which are scheduled before e in S are also scheduled before e in S'. Thus, we deduce that

$$A(e|S') = A(e|S) \geq m(S), \quad \forall e \in X \text{ and } (S(e) < S(C_u^h) \text{ or } S(e) > S(C_{v'}^{h'})). \quad (2.21)$$

Let e' be an event of X such that $S(e') \in [S(C_u^h), S(C_{v'}^{h'})]$. The minimum of $A(e'|S)$ (resp. $A(e'|S')$) is reached either at the pivot index of α or at that of β. Thus,

$$A(e'|S) \geq \sum_{\{i \notin \alpha \cup \beta | S(i) \leq S(e)\}} a_i - \max\{\Delta_\alpha^-, \Delta_\alpha^- - \Delta_\alpha^+ + \Delta_\beta^-\},$$
$$A(e'|S') \geq \sum_{\{i \notin \alpha \cup \beta | S(i) \leq S(e)\}} a_i - \max\{\Delta_\beta^-, \Delta_\beta^- - \Delta_\beta^+ + \Delta_\alpha^-\}.$$

Our objective now is to prove that

$$\max\{\Delta_\alpha^-, \Delta_\alpha^- - \Delta_\alpha^+ + \Delta_\beta^-\} \geq \max\{\Delta_\beta^-, \Delta_\beta^- - \Delta_\beta^+ + \Delta_\alpha^-\}. \quad (2.22)$$

Depending on the type of α and β, we distinguish four cases.

1. α and β are respectively OP-subchain and OC-subchain. This case is impossible because of $r(\alpha) \leq r(\beta)$.

2. α and β are OP-subchains. In this case we have $\Delta_\alpha^- \leq \Delta_\alpha^+$ and $\Delta_\beta^- \leq \Delta_\beta^+$. From the assumption $r(\alpha) \leq r(\beta)$, it follows that $\Delta_\alpha^- \geq \Delta_\beta^-$. Thus, $\Delta_\alpha^- \geq \max\{\Delta_\beta^-, \Delta_\beta^- - \Delta_\beta^+ + \Delta_\alpha^-\}$. So, the inequality (2.22) is verified.

3. α and β are OC-subchains. In this case we have $\Delta_\alpha^- > \Delta_\alpha^+$ and $\Delta_\beta^- > \Delta_\beta^+$.

From the assumption $r(\alpha) \leq r(\beta)$, it follows that $-\Delta_\alpha^+ \geq -\Delta_\beta^+$. Thus, $(\Delta_\alpha^- - \Delta_\alpha^+ + \Delta_\beta^-) \geq \max\{\Delta_\beta^-, \Delta_\beta^- - \Delta_\beta^+ + \Delta_\alpha^-\}$. So, the inequality (2.22) is verified.

4. α and β are respectively OC-subchain and OP-subchain. In this case we have $\Delta_\alpha^- > \Delta_\alpha^+$ and $\Delta_\beta^- \leq \Delta_\beta^+$. It follows that $\Delta_\alpha^+ \geq \Delta_\beta^- - \Delta_\beta^+ + \Delta_\alpha^-$ and $\Delta_\alpha^- - \Delta_\alpha^+ + \Delta_\beta^- \geq \Delta_\beta^-$. So, the inequality (2.22) is verified.

Therefore,

$$A(e'|S') \geq m(S') \geq m(S), \quad \forall e' \in X \text{ and } S(e') \in [S(C_u^h), S(C_{v'}^{h'})]. \tag{2.23}$$

From (2.21) and (2.23) it follows that $m(S') \geq m(S)$. □

Proof of Theorem 2.5. By Proposition 2.3 at least one optimal schedule is of standard form. Let consider such an optimal schedule. We repeatedly interchange, if necessary, two adjacent subchains of this schedule in order to obtain a new schedule S, which has the property that $r(\alpha) \geq r(\beta)$ for any adjacent pair of subchains α and β. By Proposition 2.3, S is also optimal. Let S' be any schedule of standard form with the property that $r(\alpha) \geq r(\beta)$ for any adjacent pair of subchains α and β. S' can be obtained from S by zero or more interchanges of adjacent subchains with the same rank. This implies $m(S') \geq m(S)$ by Proposition 2.3. Therefore, S' is also optimal. □

2.5.3 The series-parallel case

We now consider a more general case, where the precedence relations involved can be represented by a series-parallel graph. This special case of ERCPSP is an extension of the problem considered by [Abdel-wahab and Kameda, 1978], where more than one event can be executed at the same time.

A series-parallel graph $G = (X, U)$ is a directed graph which can be obtained recursively from a single node by two operations, the *series composition* (Definition 2.2) and the *parallel composition* (Definition 2.3) of two series-parallel subgraphs [Valdes et al., 1982].

Definition 2.2. *Let $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$ be two series-parallel graphs on disjoint sets. The series composition $G_s = (X_s, U_s)$ of G_1 and G_2 is defined as follows. $X_s = X_1 \cup X_2$ and $i \prec j \in U_s$ if and only if $i \prec j \in U_1 \cup U_2$, or $i \in X_1$ and $j \in X_2$. The sets X_1 and X_2 are termed the series blocks of G_s.*

Definition 2.3. *Let $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$ be two series-parallel graphs on disjoint sets. The parallel composition $G_p = (X_p, U_p)$ of G_1 and G_2 is defined as*

follows. $X_p = X_1 \cup X_2$ *and* $i \prec j \in U_p$ *if and only if* $i \prec j \in U_1 \cup U_2$. *The sets* X_1 *and* X_2 *are termed the parallel blocks of* G_p.

[Abdel-wahab and Kameda, 1978] define the series-parallel graphs as follows.

Definition 2.4. *[Abdel-wahab and Kameda, 1978] A graph is a series-parallel graph if it can be reduced to a graph consisting of only two nodes with an arc between them by a sequence of the following operations.*

1. *Replace two arcs,* (u, v) *and* (v, w), *and the node* v *by a single arc* (u, w) *if* $|\Gamma^-(v)| = |\Gamma^+(v)| = 1$.

2. *Delete an arc in parallel to another arc.*

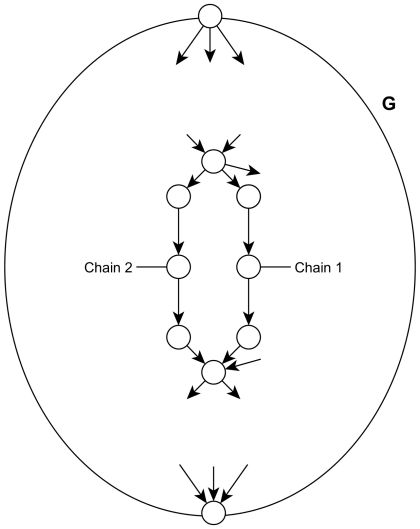


Figure 2.8 – Series-parallel graph

Form Definition 2.4, any series-parallel graph has a subgraph consisting of two chains (see Fig 2.8), unless it is a single chain. If the precedence relations are represented by a series-parallel digraph, then a total order of events can be defined as follows. We first find two parallel chains using the method proposed by [Abdel-wahab and Kameda, 1978]. Then we apply Algorithm 2.4 to obtain an optimal schedule. We replace the two parallel chains by a single chain obtained by adding an arc between two adjacent subchains according to the optimal schedule. Thus, we obtain another simpler series-parallel graph. If we continue to repeat this operation, the outcome is a single chain which corresponds to a total order of events. This method is illustrated by the example of Fig 2.9. Abdel-wahab and Kameda proved that any schedule, which respects this total order of events, is optimal for the bicretirion problem. The same proof can be used in the case of ERCPSP. It is based

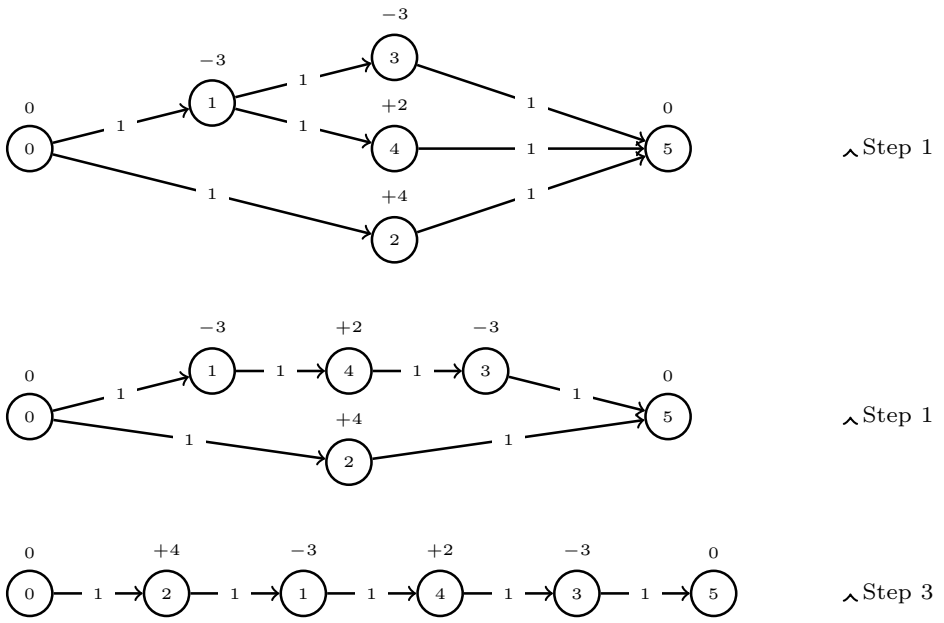


Figure 2.9 – Example

on the same principle as the case of parallel chains (The events of each subchain are clustered around the pivot event, then the subchains are merged). Moreover, the feasibility of the problem in this case can be calculated using an $O(n^2)$ algorithm [Abdel-wahab and Kameda, 1978].

2.5.4 The interval order case

In this section, we investigate another special case of ERCPSP with single resource, where the precedence graph $G = (X, U)$ is an interval order graph and the time lags are strictly positive. We introduce for this special case a list algorithm to construct feasible schedules. The priorities of events are defined using the proprieties of interval orders, so that all production events are scheduled when they are ready, and all consumption events are scheduled when they are ready and ascendant of all unscheduled production events.

2.5.4.1 Interval orders

An interval order graph $G = (X, U)$ is a directed acyclic graph, such that for each $i \in X$, one can associate a closed interval $l(i)$ in the real line, such that for all $i, j \in X$, $(i, j) \in U$ if and only if $x < y$ for all $x \in l(i)$ and $y \in l(j)$ [Papadimitriou and Yannakakis, 1979]. The system of intervals $l(i)$ is called an interval representation of G. Fig 2.10 gives an example of interval order graph and

Fig 2.11 shows the interval representation of this example.

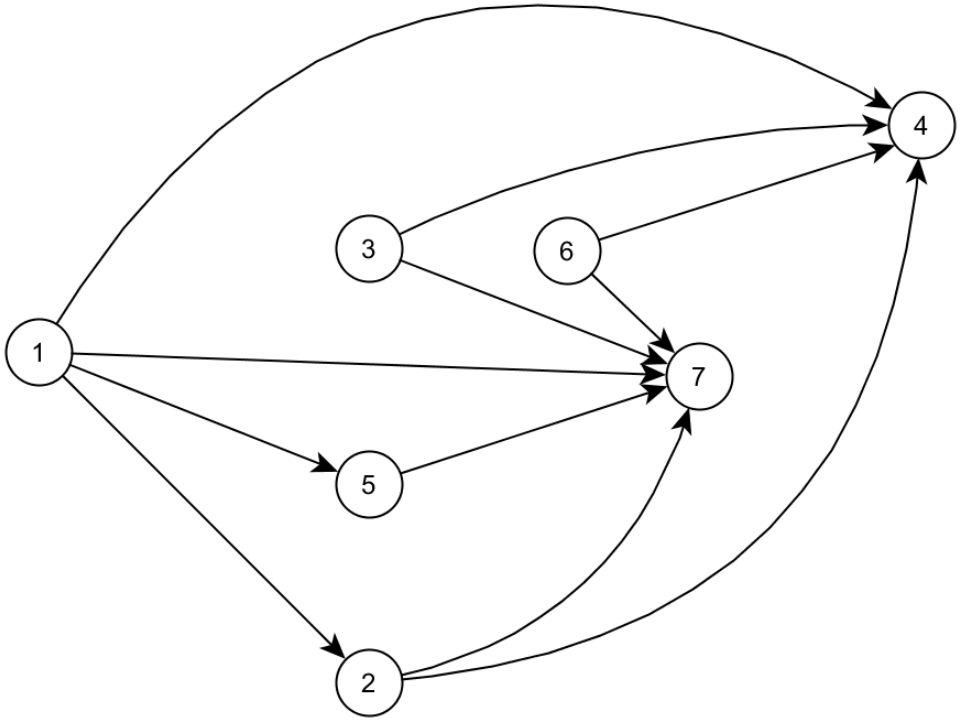


Figure 2.10 – Interval order graph

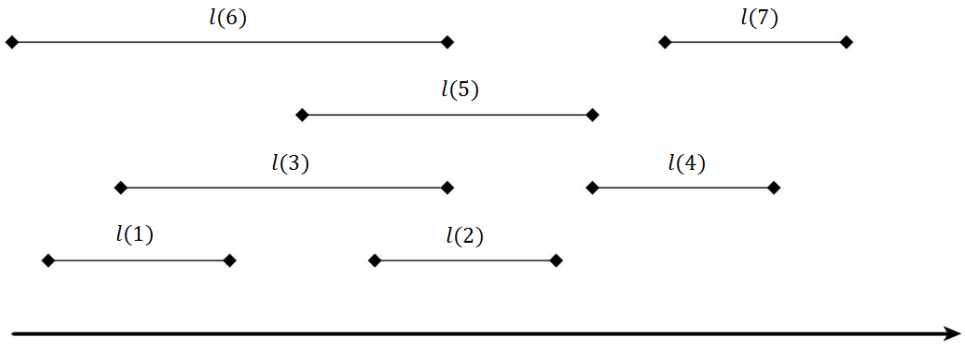


Figure 2.11 – Interval representation

Interval orders have a very nice property: the sets of successors of the events of an interval order graph form a total order.

Proposition 2.4. *[Palem and Simons, 1993] Let (X, U) be an interval order graph. Then for $i, j \in X$, either all the successors of i are also successors of j, or all the*

successors of j are also successors of i.

Proposition 2.5. *[Palem and Simons, 1993] Let (X, U) be an interval order graph. Then for $i, j \in X$, either all the predecessors of i are also predecessors of j, or all the predecessors of j are also predecessors of i.*

So, for each interval order graph we can found a total order of X $(i_0, i_1, ..., i_{n+1})$, such that $\bar{\Gamma}^+(i_{n+1}) \subseteq \bar{\Gamma}^+(i_n) \subseteq ... \subseteq \bar{\Gamma}^+(i_0)$ (the descendants of i_0 include the descendants of i_1 which include the descendants of i_2,..., which include the descendants of i_{n+1}). The idea to solve the decision problem of ERCPSP with interval order graph is to schedule the production events as soon as possible, and the consumption events when they are ready respecting the list $(i_0, i_1, ..., i_{n+1})$.

2.5.4.2 List schedule for interval order case

A list algorithm can be used to solve the decision problem of this special case of ERCPSP (see Algorithm 2.6). The structure of this algorithm is simple. In the first phase, a priority is attributed to each event. Based on this priority, the events are ordered into a list. In each step, the event with the highest priority among the ready events is chosen and added to the list. An event is said to be ready, if all its predecessors are already in the list. After adding the chosen event to the list, the new set of ready events is determined and the step is repeated until all events are contained in the list. The priorities of events are defined as follows.

- For $ep_1, ep_2 \in X^p$, ep_1 has a higher priority than ep_2 iff $ES(ep_1) \leq ES(ep_2)$.
- For $ep \in X^p, ec \in X^c$, ep has a higher priority than ec.
- For $ec_1, ec_2 \in X^c$, ec_1 has a higher priority than ec_2 iff $|\bar{\Gamma}^+(ec_1)| \geq |\bar{\Gamma}^+(ec_2)|$.

In the second phase, the algorithm iterates over the list built in the first phase and determines the occurrence time of each event. In each iteration of the algorithm, all the ready production events are executed first, then the consumption event with the highest number of successors.

Suppose that event j is scheduled just after event i. The occurrence time $S(j)$ of j is given by $S(j) = \max\{S(i), \max\{S(e) + v_{ei} \mid (e, i) \in U\}\}$. If the resource level in each iteration is nonnegative, then the obtained schedule is feasible. The algorithm terminates when a full schedule is constructed or a resource conflict is detected. Note that in Algorithm 2.6 the two phases are carried out simultaneously.

Proposition 2.6. *Let I be an instance of ERCPSP with interval order precedence graph and strictly positive time lags. I admits a solution if and only if Algorithm 2.6 terminates when a schedule is constructed.*

Algorithm 2.6: Algorithm to solve the interval order case

1. Schedule all the ready production events;
2. Schedule the consumption event with the highest number of successors;
3. **if** an over-consumption is detected **then**
   STOP without feasible Schedule;
4. **if** some events are not scheduled yet **then** Goto (1);

Proof. Suppose that Algorithm 2.6 constructs a schedule S. It is easy to verify that S satisfies all precedence and resource constraints. Thus, S is a feasible schedule for I.

Now let us consider an iteration k of the algorithm where a consumption event ec_1 is scheduled. Suppose that I admits a feasible schedule S' and the algorithm detects an over-consumption in iteration k. If S' is feasible then there exists at least one consumption event ec_2 such that $|\bar{\Gamma}^+(ec_2)| \leq |\bar{\Gamma}^+(ec_1)|$, and the set $\bar{\Gamma}^+(ec_2)$ contains at least one production event not in $\bar{\Gamma}^+(ec_1)$. This is impossible because form Proposition 2.4 we have $\bar{\Gamma}^+(ec_2) \in \bar{\Gamma}^+(ec_1)$.

□

2.6 Dynamic Programming: parallel chain case

In this section, we propose a dynamic programming algorithm for the ERCPSP with parallel chain precedence graph and single resource to minimize the makespan. We suppose that all the time lags are strictly positive.

2.6.1 Dynamic programming approach

A state in our dynamic programming approach is defined by a triplet $(\tilde{X}, t, \xi)$, where $\tilde{X} \subseteq X$ is a subset of scheduled events, $t = (t_1, t_2, ..., t_{SP})$ is a vector of occurrence times, and $\xi = (\xi_1, \xi_2, ..., \xi_{SP})$ is a vector of resource levels. Note that t_h is the occurrence time of the last event of chain h $(1 \leq h \leq SP)$ belonging to $\tilde{X}$ and ξ_h is the level of resource at t_h. A state $(\tilde{X}, t, \xi)$ is said to be feasible iff ξ_h is positive for each $h \in \{1, 2, ..., SP\}$. We denote by Θ the set of all feasible states. The makespan associated with a state $(\tilde{X}, t, \xi)$ is given by $F(\tilde{X}, t, \xi)$:

$$F(\tilde{X}, t, \xi) = \begin{cases} \max\{t_1, t_2, ..., t_{SP}\} & \text{if } \xi_i \geq 0, 1 \leq i \leq SP. \\ +\infty & \text{otherwise.} \end{cases}$$

Let $F(\tilde{X})$ be the optimal makespan of a schedule that processes all events in $\tilde{X}$.

$$F(\tilde{X}) = \min_{(\tilde{X},t,\xi)\in\Theta} F(\tilde{X},t,\xi).$$

So, the optimal makespan of the problem is given by $F(X)$. In the following, we show how we can enumerate all these feasible states.

2.6.2 State generation

The dynamic programming algorithm requires a procedure for enumerating feasible states. The enumeration procedure considers states of the form $(\tilde{X},t,\xi)$ such that $|\tilde{X}| = \tilde{n} < n$, and creates new states of the form $(\tilde{X}' = \tilde{X} \cup \{e\}, t', \xi')$ by adding potential successors with different occurrence times. Of course, all the states are generated iteratively starting from the initial state $(\{0\},(0,...,0),(0,...,0))$.

Let $(\tilde{X},t,\xi)$ be a feasible state and C_{max} be a hypothetic makespan. Let e_1 be the last event of chain h belonging to $\tilde{X}$. We suppose that e_2 is the direct successor of e_1 and $LS(e_2)$ is its latest occurrence time according to C_{max}. For each $\theta \in [t_h + v_{e_1,e_2}..LS(e_2)]$, a state $(\tilde{X}',t',\xi')$ is generated as follows:

$$\tilde{X}' = \tilde{X} \cup \{e_2\}$$

$$t'_i = \begin{cases} \theta & \text{if } i = h \\ t_i & \text{otherwise.} \end{cases}$$

$$\xi'_i = \begin{cases} \xi_i & \text{if } i \neq h \text{ and } \theta < t_i \\ \xi_i + a_{t_i} & \text{otherwise.} \end{cases}$$

Suppose that the time window length of each event is upper bounded by a constant Δ. The enumeration procedure generates in each iteration at most $SP \times \Delta^{SP}$ states. Thus, a total of $O(n^{SP} \times \Delta^{2\times SP})$ states are generated by this method, which is pseudo-polynomial.

2.7 Conclusion

We have presented the ERCPSP which is a general scheduling problem where the availability of resources is depleted and replenished. ERCPSP is a generalization of RCPSP where activities requiring renewable resources are replaced by events consuming or producing nonrenewable resources. We have shown the connection between ERCPSP and other scheduling problem with production and consumption

of resources. Moreover, we have introduced the Decision and the Resource Usage Problem of ERCPSP and we have reported some complexity results. The decision problem in its general case is NP-complete, however some special cases can be resolved in polynomial time. We have presented four polynomial cases of ERCPSP which are the relocation problem, the parallel chain case, the series-parallel case and the interval order case. Finally, we have reported a dynamic programming algorithm to solve the parallel chain case of ERCPSP.

Lower bounds for the ERCPSP

Sommaire

3.1 Introduction **53**
3.2 Classical scheduling problems and JPPS **55**
3.3 Lower bounds for ERCPSP **60**
3.4 LP-Based lower bound **71**
3.5 Computational results **73**
3.6 Conclusion **77**

3.1 Introduction

ERCPSP is a general scheduling problem where the availability of resources is depleted and replenished [Carlier et al., 2009]. An instance of ERCPSP consists of events, nonrenewable resources and generalized precedence constraints between pairs of events. Each event produces or consumes some units of resources at its occurrence time. The objective is to build a schedule that satisfies the precedence and resource constraints and minimizes the makespan.

ERCPSP is a generalization of RCPSP where activities requiring renewable resources are replaced by events consuming or producing nonrenewable resources. Some other authors have worked on models similar to ERCPSP. We can quote the works of Neumann and Schwindt [Neumann and Schwindt, 2002] and of Laborie [Laborie, 2002]. Neumann and Schwindt formalized the Project Scheduling Problem with Inventory Constraints where the availability of each resource is at any time upper and lower bounded. To solve this problem, they proposed a branch-and-bound algorithm with a filtered beam search heuristic. Laborie [Laborie, 2002] introduced the concept of a Resource Temporal Network (RTN). He proposed a constraint propagation algorithm to solve the problem. Koné et al. [Koné et al., 2013] worked on the RCPSP with Consumption and Production of Resources (RCPSP/CPR). The

particularity of this extension of RCPSP is that, in addition to renewable resources considered in the basic version, it also involves nonrenewable resources which can be consumed (or not) at the starting time of an activity in a certain amount and/or then produced in another amount at the completion time of this activity. To solve this problem, Koné et al. proposed four mixed integer linear programming models for RCPSP/CPR. ERCPSP coincides with the problem considered by [Neumann and Schwindt, 2002] and [Laborie, 2002] where no upper bound on the resource availability is prescribed.

In a recent paper Carlier et al. [Carlier et al., 2009] have generalized tools introduced for the RCPSP to the ERCPSP. For instance schedules can be built by using list algorithms. In this work, we have also been inspired by previous works on scheduling problems with renewable resource such as the Cumulative Scheduling Problem to develop new lower bounds for ERCPSP.

Lower bounds have been proposed for models similar to the ERCPSP. Neumann and Schwindt [Neumann and Schwindt, 2002] introduced two lower bounds for the Project Scheduling Problem with Inventory Constraints. One is a critical path based lower bound and the other one is similar to the Shifting Algorithm which was introduced for the Financing Problem [Carlier, 1984]. Selle [Selle, 1999] proposed a lower bound based on a time-indexed mixed-integer programming formulation and a Lagrangean relaxation of the resource constraints.

The purpose of this chapter is to introduce six lower bounds for ERCPSP. Two of them are based on the extraction of a generalized Cumulative Scheduling Problem, combined with an adapted version of Jackson's Pseudo-Preemptive Schedule [Carlier and Pinson, 2004] and the concept of energetic reasoning. Two further lower bounds respectively result from applying Carlier and Rinnooy Kan's Shifting Algorithm to a Financing Problem and iteratively testing the feasibility of associated network flow problems in a dichotomic search method. The last two lower bounds are destructive lower bounds computed using a general linear programming scheme. This linear programming scheme is based on a decomposition of the time horizon into successive intervals.

The remaining of this chapter is structured as follows. In Section 3.2 we present the Jackson's Pseudo-Preemptive Schedule and its different adaptations. In Section 3.3 we present four new lower bounds inspired by previous works on scheduling problems with renewable resource. In Section 3.4 we present a general linear programming scheme for computing a lower bound on the makespan, and we introduce two destructive lower bounds for ESPCPR based on it. In Section 3.5 we report on experimental results and comment on the practical efficiency of our lower bounds, and finally we conclude this work in Section 3.6.

3.2 Classical scheduling problems and JPPS

The Jackson's Pseudo-Preemptive Schedule was initially introduced for the m-Machine Scheduling Problem [Carlier and Pinson, 1998]. Then, it was adapted to the Cumulative Scheduling Problem by Carlier and Pinson [Carlier and Pinson, 2004]. In this section, we propose to use JPPS for bounding the makespan of the Generalized Cumulative Scheduling Problem with similar complexities.

3.2.1 m-Machine Scheduling Problem

The m-Machine Scheduling Problem is denoted as $Pm/r_i, q_i/C_{\max}$. In this problem, a set Y of n activities has to be scheduled without preemption on m identical machines in order to minimize the makespan. Each activity i has a release date (or head) r_i, a processing time p_i, and a tail q_i. Tail q_i is the latency between the completion of activity i and the completion of the project. Table 3.1 provides parameters of an instance of $Pm/r_i, q_i/C_{max}$ having $n = 5$ activities and $m = 2$ machines. The associated instance of ERCPSP is given in Fig. 3.1.

Activity	1	2	3	4	5
r_i	0	0	0	4	6
p_i	4	6	7	6	5
q_i	6	5	0	0	0

Tableau 3.1 – An instance of the m-machine scheduling problem where m=2

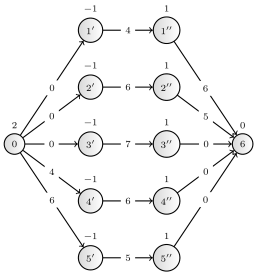


Figure 3.1 – The associated ERCPSP

3.2.2 Cumulative Scheduling Problem

In the Cumulative Scheduling Problem (CuSP), a set Y of n activities have to be scheduled without preemption using m available resource units. The aim is to minimize the makespan. Each activity i requires a constant amount e_i of resource throughout its processing and has a release date r_i, a processing time p_i, and a tail q_i. Table 3.2 gives an instance of CuSP with $n = 4$ activities and 3 resource units. Fig. 3.2 reports the associated instance of ERCPSP without negative arcs.

For an instance of CuSP, the Decision Problem is determining whether it has a feasible schedule with makespan equal to a given C_{max}. We denote this problem CuSP$/C_{max}$. It is obviously NP-complete, since the optimization problem of CuSP is NP-hard [Carlier and Pinson, 2004].

Name	r_i	q_i	p_i	e_i
A_1	0	4	3	2
A_2	0	0	6	1
A_3	3	0	2	1
A_4	3	0	4	2

Tableau 3.2 – An instance of CuSP

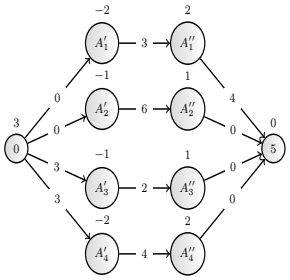


Figure 3.2 – The associated ERCPSP

3.2.3 Generalized Cumulative Scheduling Problem

For the generalized version of CuSP (GCuSP), the resource availability can vary during the project. The changes of resource availability are represented by a set of dates $\{u_1, u_2, ..., u_s\}$ and a set of quantities $\{b_1, b_2, ..., b_s\}$, where b_i is the quantity of resource that becomes available or unavailable at time u_i. If $b_i \geq 0$, then b_i resource units become available, whereas if $b_i < 0$, $|b_i|$ resource units become unavailable.

Name	r_i	p_i	q_i	e_i
A_1	2	4	1	2
A_2	0	4	3	3

Tableau 3.3 – An instance of GCuSP such that $u_1 = 0$, $u_2 = 3$, $b_1 = 6$ and $b_2 = -3$

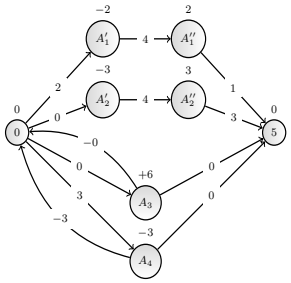


Figure 3.3 – The associated ERCPSP

In ERCPSP, an arrival (resp. departure) of b_i resources at time u_i can be represented by a production (resp. consumption) event i where $a_i = b_i$, $v_{0,i} = u_i$, $v_{i,0} = -u_i$ and $v_{i,n+1} = 0$. Table 3.3 gives an example of GCuSP with two activities, one arrival of resource at time $u_1 = 0$ and one departure of resource at time $u_2 = 3$. Fig. 3.3 shows the associated instance of ERCPSP.

3.2.4 JPPS

JPPS was introduced by Carlier and Pinson [Carlier and Pinson, 1998] for the m-machine scheduling problem. In a pseudo-preemptive schedule, the preemption of any available activity is allowed. We also assume that a machine can be shared by a group of activities and that an activity can be processed on more than one machine at a time. So, the number of machines assigned to an available activity i at time t, denoted by $\alpha_i(t)$, is not necessarily an integer. For building JPPS, we use a list

algorithm whose priority dispatching rule is the complete tail $c_i(t) = q_i + a_i(t)$, where $a_i(t)$ is the remaining processing time of activity i at the current time t in the list algorithm. So, the priority attached to an available activity is not fixed over the time, but depends on its residual duration. The only restriction is that at any time t, we must have $a_i(t) \geq p_i - (t - r_i)$ for any activity i. An activity is said to be partially available if $a_i(t) = p_i - (t - r_i)$: such an activity can only be scheduled at a rate $\alpha_i(t) \leq 1$. Indeed, in this case, we have $p_i - a_i(t) = t - r_i$ and the part of activity i processed in time interval $[r_i; t]$ is as large as possible. It is said to be totally available if $a_i(t) > p_i - (t - r_i)$: such an activity can be processed at a rate $\alpha_i(t) \leq m$. Thus, JPPS schedules first the inactive activities with maximal complete tail at a maximal rate consistent with their status (partially or totally available). JPPS is then composed of consecutive schedule blocks during which the subset of in-process activities and the associated rates are invariant, a schedule block B being partitioned into a set of partially available activities P and a set of totally available activities T. A schedule block starting at time t is completed at time $t + \theta$, called decision time, and associated with some event which leads to modifications on its structure. In such a block, activities of T are scheduled at the same rate α_T and those of P are processed at rate 1. The activities of the block are processed in $]t; t + \theta]$. Fig. 3.4 shows the JPPS built on the instance of Table 3.1.

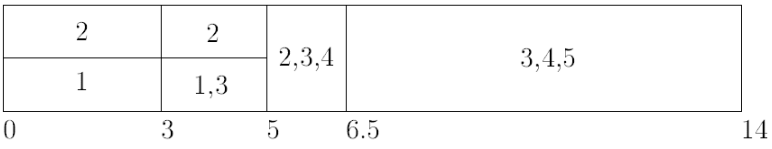


Figure 3.4 – Solution of JPPS

Theorem 3.1. ***[Carlier and Pinson, 1998]***

$$C(JPPS) = max\{max_{i \in Y}(r_i + p_i + q_i), max_{(J \subseteq Y, |J| \geq m)} G'(J)\}$$

$C(JPPS)$ denotes the makespan of JPPS and $J \subseteq Y$ denotes a subset of activities such as $|J| \geq m$, and $G'(J)$ is defined by:

$$G'(J) = \frac{1}{m}(r_{i_1} + ... + r_{i_m}) + \frac{1}{m}\sum_{i \in J} p_i + \frac{1}{m}(q_{j_1} + ... + q_{j_m}),$$

where $i_1 ... i_m$ (resp. $j_1 ... j_m$) denote the m first activities in J rearranged in a nondecreasing order of heads (resp. tails).

Note that $max_{i \in Y}(r_i + p_i + q_i)$ corresponds to the length of the critical path, and $G'(J)$ is a lower bound on the makespan of $Pm/r_i, q_i/C_{max}$, which takes into account heads and tails associated with activities belonging to J and the available machines.

3.2.4.1 Adaptation of JPPS to CuSP

Carlier and Pinson [Carlier and Pinson, 2004] have proposed two methods to adapt JPPS to CuSP. The first one is to associate with each activity i, e_i activities requiring one machine. Then JPPS is applied on the derived instance involving $\sum e_i$ activities. This method introduces a pseudo-polynomial component in the complexity associated with JPPS construction. However, in the corresponding schedule, each activity derived from i is executed during the same periods with the same rates. So, a better idea is to modify rules defining rates and schedule blocks without adding additional activities (see the proposition bellow). Consequently, JPPS can be computed for the CuSP with the same complexity as $Pm/r_i, q_i/C_{max}$ ($e_i = 1; \forall i \in I$).

Proposition 3.1. *Let t and t' denote two consecutive decision times in JPPS, and $B = P \cup T$ the schedule block starting at time t, where P is the set of partially available activities and T the set of totally available activities with maximal complete tail c_T. For any time $u \in]t; t']$, the activities of P are scheduled at rate 1 and the activities of T are scheduled at rate*

$$\alpha = \frac{m - \sum_{i \in P} e_i}{\sum_{i \in T} e_i}$$

In order to apply JPPS to the example of Table 3.2 with the first method, we create the derived instance from the first instance as follows. We associate with activity A_1 which needs two resource units, two activities A_1^1, A_1^2 requiring each of them one machine. Similarly, we associate with activity A_2 one activity A_2^1, with activity A_3 one activity A_3^1 and with activity A_4 two activities A_4^1, A_4^2. The result of JPPS is shown in Fig. 3.5. As we can see, there are four blocks in this schedule. At time 0, three activities are available: A_1^1, A_1^2 and A_2^1. They are partially available so we schedule them at rates $\alpha = 1$. At time 3, activities A_3^1, A_4^1 and A_4^2 become partially available and activities A_1^1 and A_1^2 are finished, but the priority of activity A_3^1 is smaller than others, so B $= \{A_2^1, A_4^1, A_4^2\}$, T $= \{A_2^1\}$ and P $= \{A_4^1, A_4^2\}$. We schedule activities A_4^1 and A_4^2 at rate 1 and A_2^1 at rate $\alpha_T = 1$. Until time 4 the priority of activity A_3^1 becomes equal to the priority of activity A_2^1. So we have B $=$ T $\cup$ P, P $= \{A_4^1, A_4^2\}$, T$=\{A_2^1, A_3^1\}$, we schedule activities A_4^1 and A_4^2 at rate $\alpha = 1$

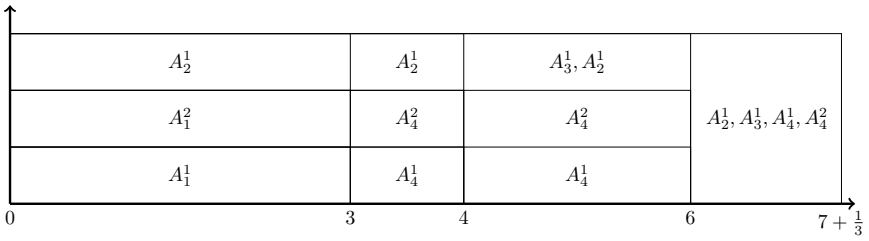


Figure 3.5 – Solution of JPPS with the first method

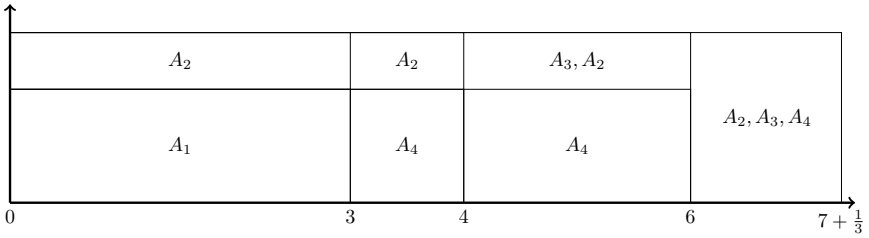


Figure 3.6 – Solution of JPPS with the second method

and activities A_2^1, A_3^1 at rate $\alpha_T = \frac{1}{2}$. At time 6, all four activities have the same priority, so we put them in the same schedule block at rate $\alpha = \frac{3}{4}$. All of them finish at time $7 + \frac{1}{3}$. Fig. 3.6 shows the result of JPPS adapted to CuSP using the second method. As we can see, this method gives the same result as the first one but with much less activities.

3.2.4.2 Adaptation of JPPS to GCuSP (GJPPS)

In JPPS, the current schedule block can be modified by the following events:

(E_1) A not in-process activity becomes available.

(E_2) An in-process activity is completed.

(E_3) A not in-process available activity enters into the process.

(E_4) A totally available activity becomes partially available.

(E_5) A partially available activity becomes totally available.

To adapt JPPS to GCuSP, we add a new type of decision times not introduced in [Carlier and Pinson, 1998]. These new decision times correspond to the moments of resource availability changes. Fig. 3.7 shows the application details of the GJPPS algorithm on the example of Table 3.3. At time 0, we have an arrival of 6 units of resource. Activity A_2 becomes partially available so we schedule it at rate 1. At time

2, A_1 is partially available. As A_1 and A_2 are both partially available we schedule them at rate 1. At time 3, we have a departure of 3 units of resource and A_1, A_2 become totally available. So, we schedule them at rate $\alpha_T = \frac{3}{5}$. At time $4 + \frac{2}{3}$, A_2 completes its execution and A_1 remains totally available. So, we schedule it at rate $\alpha_T = \frac{3}{2}$. Finally we obtain a lower bound equal to $7 + \frac{2}{3}$.

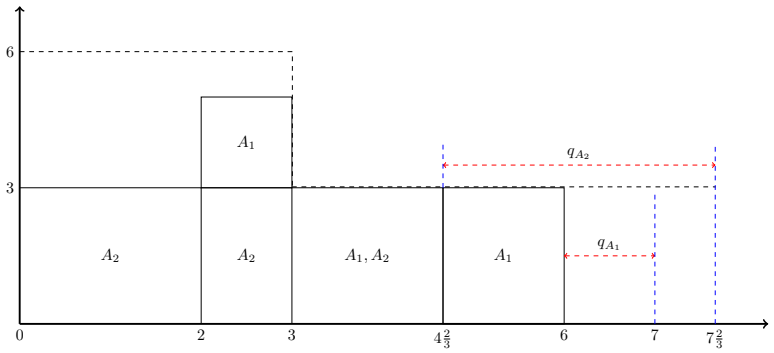


Figure 3.7 – Solution of GJPPS for an instance of GCuSP

3.3 Lower bounds for ERCPSP

In this section, we present four destructive lower bounds for ERCPSP. Without loss of generality, we suppose that there is only one type of resource. For the case of multiple resources, the lower bound is then the maximum among all the lower bounds calculated for each single resource.

3.3.1 Notation

Given an instance $I = (X, U, a, v)$ of ERCPSP, the set of events can be separated into two subsets: one that contains all production events and one with all consumption events. Let X^p be the set of all production events and X^c the set of all consumption events. If path of nonnegative length between two events i and j exists in the graph (X, U), then we denote by $l_{i,j}$ the length of the longest path linking i to j. The earliest occurrence time of event i is denoted by ES_i which is equal to $l_{0,i}$. The latest occurrence time of event i according to a given hypothetic makespan C_{max} is denoted by $LS_i(C_{max})$ which is equal to $C_{max} - l_{i,n+1}$.

3.3.2 Lower bound based on JPPS

We compute a lower bound for ERCPSP by associating an instance of transportation problem with an instance of ERCPSP. Then we solve the transportation problem and

we associate with its solution an instance of the Generalized Cumulative Scheduling Problem (GCuSP). Finally, we apply GJPPS to obtain a lower bound for the instance of GCuSP which is considered as a lower bound for ERCPSP.

3.3.2.1 Transportation problem associated with ERCPSP

For an instance of ERCPSP denoted by $I = (X, U, a, v)$, we establish a bipartite graph $\tilde{G} = (X^c \cup X^p, \tilde{U})$, where $\tilde{U} = \{(ec, ep)/ec \in X^c, ep \in X^p\}$. For each arc $u = (ec, ep) \in \tilde{U}$, profit $\gamma_{ec,ep}$ is equal to $l_{ec,ep}$ if a positive path from ec to ep exists in $G = (X, U)$, otherwise $\gamma_{ec,ep} = -\infty$. Let a^-_{ec} be the quantity of resource consumed by consumption event ec ($a^-_{ec} = -a_{ec}$) and a^+_{ep} be the quantity of resource produced by production event ep ($a^+_{ep} = +a_{ep}$). An instance of the transportation problem $(X^c \cup X^p, \tilde{U}, a^-, a^+, \gamma)$ is established.

The interest of associating with ERCPSP a transportation problem is to get a relaxation for ERCPSP based on GCuSP, so we can use existing methods to calculate a lower bound. An activity in the GCuSP is composed of two events in the ERCPSP, where a consumption event ec is followed by a production event ep. The processing time of this activity is equal to the length of the longest path from ec to ep and the resource requirement is set to be the resource flow from ec to ep, which is denoted as $f_{ec,ep}$. In order to tighten the lower bound given by the GJPPS, we must maximize the total energy required by all activities, where the energy required by an activity is equal to its processing time multiplied by its resource requirement. This case can be treated by the transportation problem which can be formulated as a linear programming problem as follows:

Maximize:

$$\sum_{(ec,ep)\in\tilde{U}} \gamma_{ec,ep} \times f_{ec,ep}$$

Subject to constraint:

$$\sum_{ec\in X^c} f_{ec,ep} \leq a^+_{ep} \qquad \forall ep \in X^p$$

$$\sum_{ep\in X^p} f_{ec,ep} \leq a^-_{ec} \qquad \forall ec \in X^c$$

$$f_{ec,ep} \geq 0 \qquad \forall (ec, ep) \in \tilde{U}$$

The objective of solving this model is to determine the unknown $f_{ec,ep}$ that maximizes the total transportation profit while satisfying all supply and demand constraints.

Finding an optimal solution increases the computation time of the algorithm and

does not ensure a better lower bound. Thus, we focus on heuristics. There are several heuristics which are effective and have good performance. We quote for example the Row Minimum Method ($O(n^2)$), the Column Minimum Method ($O(n^2)$) [Gass, 2003] and the Matrix Minimum Method ($O(n^2 \log n)$) [Gass, 2003, Ramamurthy, 2007].

3.3.2.2 Transformation from ERCPSP into GCuSP

Once the flow is computed either optimally or heuristically, we can transform ERCPSP into GCuSP, where each assignment of resource in the flow is considered as an activity. The resource flow between two events ec and ep is converted into the resource required by the activity and $l_{ec,ep}$ is converted into the duration of the activity. For each consumption event ec, we compute the length of the longest path from the beginning of project to ec and store it as r_{ec}. For each production event ep, we compute the length of the longest path from ep to the end of the project and store it as q_{ep}. Therefore, for an activity which is composed of a consumption event ec and a production event ep, r_{ec} is its release date and q_{ep} its latency duration or tail.

The changes of resource availability are provided by the production and consumption events which are incompletely covered by the flow. Let $\bar{X} = \bar{X}^c \cup \bar{X}^p$ be the set of these events, and let $\bar{a}_i$ be the remaining resource units to be produced or consumed by event i.

$$\begin{cases} \bar{X}^c = \{ec \mid ec \in X^c, \quad \bar{a}_{ec} = a_{ec} + \sum_{ep \in X^p} f_{ec,ep} < 0\} \\ \bar{X}^p = \{ep \mid ep \in X^p, \quad \bar{a}_{ep} = a_{ep} - \sum_{ec \in X^c} f_{ec,ep} > 0\} \end{cases}$$

To determine all dates when changes of resource availability occur, we have to fix a date u_i when $b_i = \bar{a}_i$ units of resource become available or unavailable for each incompletely covered event i. Since we do not know the project duration, it is not so easy to determine the dates of consumption. Thus, relaxations of consumption of resources are needed. The easiest way is to fix all production events at their earliest occurrence times and to ignore all consumption events.

Once the relaxation is made, we apply GJPPS to the corresponding GCuSP. The obtained lower bound is taken as a constructive lower bound for ERCPSP. The complexity of this method is $O(n^3)$. Indeed, we have to compute a matrix of longest paths ($O(n^3)$), and to determine a solution of the transportation problem ($O(n^2 \log n)$). In the next section we report a method to improve this bound by defining the improved JPPS which is a destructive lower bound.

3.3.2.3 Improved JPPS Algorithm

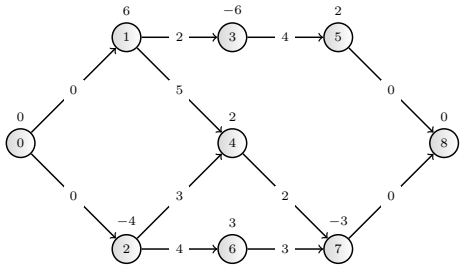
Since we have omitted several consumption events, the lower bound is loose. In order to improve it, we introduce the improved JPPS (IJPPS) in which we fix production events at their earliest occurrence times and consumption events at their latest occurrence times. But this needs a hypothetic duration C_{max} of the project in order to calculate them. For an instance G of GCuSP, let $GJPPS(G)$ denote the lower bound calculated by GJPPS. Calculation details are provided in Algorithm 3.1.

Algorithm 3.1: Improved JPPS algorithm.

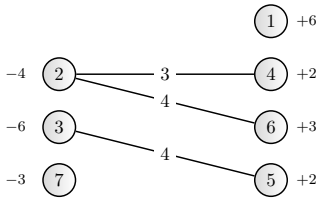
Input: An instance I of ERCPSP.
Output: Destructive bound $IJPPS$.

i) Associate with I an instance G of GCuSP;
ii) Set the dates of resource arrivals of G by setting the incompletely covered production events at their earliest occurrence times ($u_j \leftarrow ES_j$, for all $j \in \bar{X}^p$);
iii) $C_{\max} \leftarrow GJPPS(G)$ without considering any resource departures;
iv) Set the dates of resource departures of G by setting the consumption events at their latest occurrence times ($u_j \leftarrow \max\{u_j, LS_j(C_{max})\}$, for all $j \in \bar{X}^c$);
v) Recalculate $C \leftarrow GJPPS(G)$ considering all resource departures;
vi) **if** $C > C_{\max}$ **then** $C_{\max} \leftarrow C_{\max} + 1$ and goto iv) ;
$IJPPS \leftarrow C_{\max}$;
return $IJPPS$

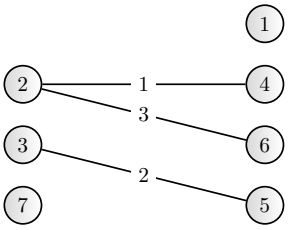
Fig. 3.8(a) gives an example of scheduling problem with production and consumption with $n = 7$ events. Events 1, 4, 5 and 6 produce respectively 6, 2, 2 and 3 units of resource and events 2, 3 and 7 consume respectively 4, 6 and 3 units of resource. Thus, we get $X^c = \{2, 3, 7\}$ and $X^p = \{1, 4, 5, 6\}$. Then for each pair of $c \in X^c$ and $ep \in X^p$, if there exists a positive path from ec to ep in the precedence graph, we calculate profit $\gamma_{ec,ep} = l_{ec,ep}$. A transportation problem is associated with the ERCPSP instance as shown in Fig. 3.8(b), where the values on arcs represent their profits. If we solve it using the Matrix Minimum heuristic, we obtain a flow as the one shown in Fig. 3.8(c), where $f_{2,4} = 1$, $f_{2,6} = 3$ and $f_{3,5} = 2$. Each flow $f_{ec,ep}$ can be considered as an activity in the GCuSP. For the problem above, we introduce three activities $(2, 4)$, $(2, 6)$ and $(3, 5)$. The arrivals of resources are given by events 1 and 4. The departures of resources are given by events 3 and 7. The associated GCuSP instance is presented in Fig. 3.8(d). If we apply the improved JPPS on this instance, we get a lower bound equal to 8. By comparing this lower bound with the length of critical path which is equal to 7, we get an improvement of 1.



(a) ERCPSP



(b) a transportation problem associated with the example



(c) optimal solution of the transportation problem

Name	r_i	q_i	p_i	e_i
activity (2,4)	0	2	3	1
activity (2,6)	0	3	4	3
activity (3,5)	2	0	4	2

	event (1)	event (4)	event (3)	event (7)
u_i	0	5	*	*
b_i	+6	+1	-4	-3

(d) GCuSP instance associated with the example

Figure 3.8 – An example: From ERCPSP to GCuSP

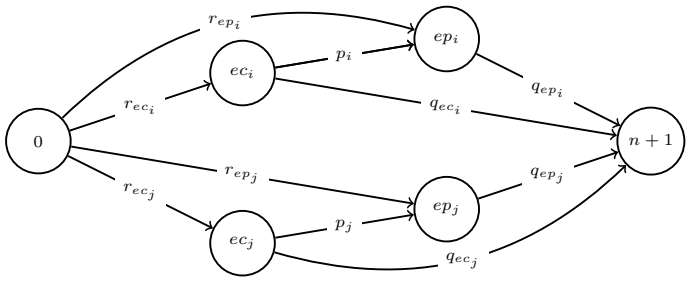
The evaluation based on the GCuSP instance can be improved by introducing several instances of GCuSP. Let us return to the graph of the transportation problem. We have $2n'$ events $ec_1, ec_2, ..., ec_{n'}$, $ep_1, ep_2, ..., ep_{n'}$. These events are associated in pairs to form activities. For instance (ec_1, ep_1) corresponds to activity 1 , ..., $(ec_{n'}, ep_{n'})$ to activity n'.

Let us choose a subset J of these activities and introduce GCuSP(J) as follows. For each activity $i \in J$, we replace its release date by $\max(r_{ec_i}, l_{0,ep_i} - p_i)$. For each activity $i \in \bar{J}$, we replace its tail by $\max(q_{ep_i}, l_{ec_i,n+1} - p_i)$ where $\bar{J}$ is the absolute complement of J.

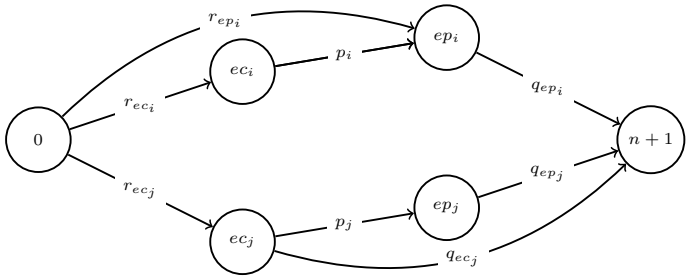
Proposition 3.2. *The optimal makespan of GCuSP(J) is a lower bound on the minimum makespan of the initial ERCPSP.*

Proof. Fig. 3.9(a) shows an instance I of ERCPSP. Fig. 3.9(b) represents a relaxation of I obtained by removing the arc valued by q_{ec_i} for i belonging to J and the arcs valued by r_{ep_i} for i belonging to $\bar{J}$. We consider a solution S of the relaxed instance. Then we change this solution by shifting right (resp. left) event ec_i for $i \in J$ (resp. ep_i for $i \in \bar{J}$). Let S' denote the new solution of the relaxed instance:

$$S'(ec_i) = \begin{cases} \max(S(ec_i), S(ep_i) - p_i) & \text{if } i \in J, \\ S(ec_i) & \text{if } i \in \bar{J} \end{cases}$$



(a) An instance I of ERCPSP.



(b) A relaxation of I.

Figure 3.9 – Proof of Proposition 3.2

$$S'(ep_i) = \begin{cases} S(ep_i) & \text{if } i \in J, \\ \max(S(ep_i), S(ec_i) + p_i) & \text{if } i \in \bar{J} \end{cases}$$

Therefore, we obtain a new solution of the relaxed instance which is also a solution of GCuSP(J). Q.E.D □

3.3.3 Extension of Energetic Reasoning

Erschler et al. [Erschler et al., 1991] and Lopez et al. [Lopez et al., 1992] developed the energetic reasoning to solve the CuSP, they were inspired by the work of Lahrichi [Lahrichi, 1982]. The energetic approach has been formalized and evaluated from a theoretical as well as an experimental point of view by Baptiste et al. [Baptiste et al., 1999]. Since its initial development for CuSP, the energetic reasoning has been used for solving more complex scheduling problems including RCPSP [Baptiste et al., 1999]. In this section, we show how we can extend it for the ERCPSP.

To use the energetic reasoning, we need to get a relaxation for ERCPSP based on CuSP/C_{max} where C_{max} is a hypothetic makespan. Thus, we follow the same process to get an instance of GCuSP as explained in the previous section while transforming the unassigned production and consumption events $\bar{X}^p \cup \bar{X}^c$ into

activities. Therefore, for an instance of ERCPSP, we associate an instance of the transportation problem. Then, we solve it and we associate its solution with an instance of CuSP/C_{max} where each assignment of the resource is regarded as an activity.

Production and consumption events $\bar{X}^p \cup \bar{X}^c$, which are incompletely covered by the flow, are transformed into activities as follows. For each production event ep of $\bar{X}^p$, we introduce an activity i with release date $r_i = 0$, processing time $p_i = ES_{ep}$, tail $q_i = C_{max} - ES_{ep}$ and resource capacity requirement $e_i = \bar{a}_{ep}$. For each consumption event ec of $\bar{X}^c$, we introduce an activity j with release date $r_j = LS_{ec}(C_{max})$, processing time $p_j = C_{max} - LS_{ec}(C_{max})$, tail $q_j = 0$ and resource capacity requirement $e_j = -\bar{a}_{ec}$. The resource availability of this new instance is equal to $\sum_{ep \in \bar{X}^p} \bar{a}_{ep}$.

Once the relaxation is made, we apply energetic reasoning to the corresponding CuSP/C_{max} in order to compute a lower bound. If the obtained bound is strictly larger than C_{max}, we deduce that this instance is infeasible and $C_{max} + 1$ is a valid lower bound for ERCPSP. Of course the complexity of this method is $O(n^3)$, even if there is no time bound adjustment [Baptiste et al., 1999].

3.3.4 Lower bound based on the Shifting Algorithm

The financing problem aims to model the financing of some project being realized. This problem is a special case of ERCPSP, where the dates of production events are fixed. It can be solved using the shifting algorithm (Section 1.4.1) in polynomial time ($O(n \log n)$). Hence, to compute a lower bound for ERCPSP, first we relax the ERCPSP to the Financing Problem by setting the production events at their earliest occurrence times and the consumption events at their latest occurrence times according to the minimum duration of the project $l_{0,n+1}$. Then, we apply the shifting algorithm to the corresponding instance. Finally, we take the makespan of the financing problem instance as a lower bound for ERCPSP. We can compute the same bound using a destructive approach as follows. Given a trial value C_{max}, we set the production events at their earliest occurrence times and the consumption events at their latest occurrence times according to C_{max}. If a conflict of resource is detected then $C_{max} + 1$ is a new valid lower bound [Carlier, 1984] [Neumann and Schwindt, 2002].

3.3.4.1 Shifting Algorithm for ERCPSP

For an instance $I = (X, U, a, v)$ of ERCPSP, we compute the earliest occurrence time of production events and the latest occurrence time of consumption events

according to $l_{0,n+1}$. We suppose that the production (resp. consumption) events are indexed in a nondecreasing order of their ES_i (resp. $LS_i(l_{0,n+1})$).

Let us denote $\Gamma = \{\tau_i = ES_{ep_i}/ep_i \in X^p\}$ the set of the earliest occurrence times and $F = \{f_i = LS_{ec_i}(l_{0,n+1})/ec_i \in X^c\}$ the set of the latest occurrence times. At each time τ_i of Γ (resp. f_i of F), $a^+_{ep_i}$ (resp. $a^-_{ec_i}$) units of resource are produced (resp. consumed).

In the shifting algorithm, the latest occurrence time of consumption events is shifted in order to satisfy the condition of resource-feasibility (Calculation details are given in Algorithm 3.2). We say that a time-feasible schedule $S = \{S(ec)/ec \in X^c\}$ is resource-feasible if the following condition is satisfied at each time t:

$$R(t) = \sum_{\{ec \mid ec \in X^c,\ S(ec) \leq t\}} a^-_{ec} \leq A(t) = \sum_{\{ep \mid ep \in X^p,\ ES_{ep} \leq t\}} a^+_{ep}$$

Algorithm 3.2: The shifting algorithm

```
begin
    if (∑_{ep∈X^p} a^+_ep < ∑_{ec∈X^c} a^-_ec) then
        There is no feasible schedule
    else
        Γ = {τ_i = ES_{ep_i}/ep_i ∈ X^p};
        F = {f_i = LS_{ec_i}(l_{0,n+1})/ec_i ∈ X^c};
        A(τ_0) := a^+_{ep_0};
        for i := 1 to |X^p| − 1 do
            A(τ_i) := A(τ_{i−1}) + a^+_{ep_i}
        μ := 0; δ := 0; R := 0;
        for i := 0 to |X^c| − 1 do
            R := R + a^-_{ec_i};
            while A(τ_μ) < R do
                μ := μ + 1
            δ := max(δ, τ_μ − f_i)
```

Fig. 3.10 shows the result of the shifting algorithm applied to the example of Fig. 3.8(a). As we can see, the algorithm computes the latest schedule of consumption events $F = \{f_0 = 0, f_1 = 3, f_2 = 7, f_3 = 7\}$ and the earliest schedule of production events $\Gamma = \{\tau_0 = 0, \tau_1 = 0, \tau_2 = 4, \tau_3 = 5, \tau_4 = 6\}$. Next, it determines δ which takes the smallest value such that $A(f_i + \delta) \geq R(S(ec_i))$ for any consumption event ec_i. Therefore, $\delta = 2$ and the optimal schedule built by the algorithm is $S(2) = 2, S(3) = 5, S(7) = 9, S(8) = 9$. So, the lower bound obtained is equal to 9. Comparing with the length of critical path which is equal to 7, we have an improvement of lower

bound by 2. It can be proved that 9 is the optimal makespan.

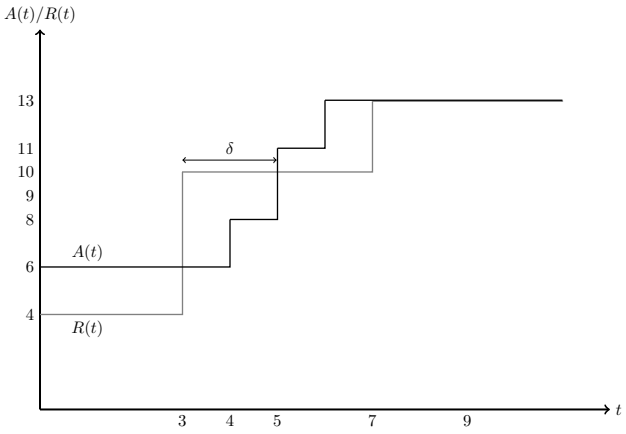


Figure 3.10 – Applying the shifting algorithm

3.3.5 Lower bound based on Network Flows

In this subsection, we present a destructive bound that we compute as follows. With an instance of ERCPSP, we associate an instance of the network flow problem by setting the value of C_{max}. If the network flow problem does not admit any solution, then $C_{max} + 1$ is a lower bound.

3.3.5.1 Network Flow problem associated with ERCPSP

Let $I = (X, U, a, v)$ be an instance of ERCPSP. For this instance we introduce a bipartite graph $G_I = (X^p \cup X^c, U_I)$. Given a trial value C_{max}, we set the production events at their earliest occurrence times and the consumption events at their latest occurrence times according to C_{max}. We consider an arc between a production event ep and a consumption event ec, if event ep can start before ec. This is obviously impossible if there exists a strictly positive path from ec to ep or if the earliest occurrence time of ep is strictly larger than the latest occurrence time of ec ($ES_{ep} > LS_{ec}(C_{max})$). If the network flow problem defined by the bipartite graph G_I does not admit any solution then $C_{max}+1$ is a lower bound for the instance I. The complexity of the method is $O(n^3)$ since we have to compute a solution of the Network Flow instance.

Fig. 3.11 shows the Network Flow problem associated with the instance of Fig. 3.8(a) when we set C_{max} to 8. As this network flow doses not admit any solution, 9 is a new lower bound. Comparing with the length of critical path, we have an improvement of the lower bound by 2.

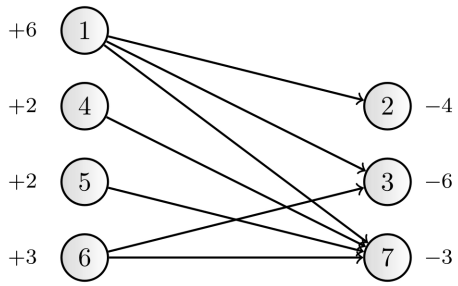


Figure 3.11 – The associated network flow

Algorithm 3.3: Destructive bound based on network flows

Input: An instance $I = (X, U, a, v)$ of ERCPSP
Output: Destructive lower bound $FLOW$

i) $UB = \sum_{i=1}^{n} \max_{(i,j)\in U}\{v_{ij}\}$;
ii) $LB =$ Length of critical path ES_{n+1} ;
iii) $Val = \lceil (LB + UB)/2 \rceil$;
iv)Generate the bipartite graph $G_I = (X^p \cup X^c, U_I)$ by setting $C_{max} = Val$;
if *the network flow defined by* G_I *admits a solution* **then**
  $UB = Val$;
else
  $LB = Val + 1$;
if $LB < UB$ **then**
  goto iii)
else if $LB > UB$ **then**
  write(Infeasible instance)
$FLOW = Val$;
return $FLOW$

In order to improve this lower bound we apply Algorithm 3.3. This algorithm takes as an input an instance of ERCPSP and makes a dichotomic search on the maximal value of C_{max}. Besides the improvement of the bound, this algorithm can detect the infeasible instances. In fact, at each iteration, the algorithm computes an upper bound UB and a lower bound LB, if $UB < LB$ we can deduce that the instance is infeasible.

3.3.6 Method to improve lower bounds

In this subsection, we present a method to improve our lower bounds by adding new precedence constraints. This method consists in deriving sufficient conditions to prove that no feasible schedule can exist with specific order of a couple of events.

Given an instance $I = (X, U, a, v)$ of ERCPSP, we define the so-called distance matrix $(Dist_{i,j}(I))_{i,j \in X}$ (i.e., the transitive hull of the time lags $l_{i,j}$, including the time lag $l_{n+1,0} = C_{max}$). Let ep be a production event and ec be a consumption event such that $(ec, ep) \notin U$ and $(ep, ec) \notin U$. We start by fixing the execution of ep before (resp. after) ec. If an infeasibility is detected, then no feasible solution can exist in which ep is scheduled before (resp. after) ec. In order to fix the execution of ep before (resp. after) ec, we need to add an arc from ep to ec (resp. ec to ep) valued by 0 (resp. 1). Let $I^1 = (X, U \cup \{(ep, ec)\}, a, v)$ and $I^2 = (X, U \cup \{(ec, ep)\}, a, v)$ two instances of ERCPSP obtained after adding these arcs. An infeasibility is detected on I^1 (resp. I^2), if the graph $(X, U \cup \{(ep, ec)\})$ (resp. $(X, U \cup \{(ep, ec)\})$) contains a strictly positive cycle, which is equivalent to the inequalities $Dist_{ep,ep}(I^1) > 0$ or $Dist_{ec,ec}(I^1) > 0$ (resp. $Dist_{ep,ep}(I^2) > 0$ or $Dist_{ec,ec}(I^2) > 0$). It is also detected if we compute a lower bound $LB(I^1)$ (resp. $LB(I^2)$) strictly larger than a given upper bound UB of the project.

Algorithm 3.4: Algorithm to improve LB

Input: An instance $I = (X, U, a, v)$
begin
  $UB = \sum_{i=1}^{n} \max_{(i,j) \in U} \{v_{ij}\}$;
  for ***all*** $(ep, ec) \in X^p \times X^c$ **do**
    if $(ep, ec) \notin U$ *and* $(ec, ep) \notin U$ **then**
      Define $I^1 = (X, U \cup (ep, ec), a, v)$ with $v_{ep,ec} = 0$;
      Define $I^2 = (X, U \cup (ec, ep), a, v)$ with $v_{ec,ep} = 1$;
      if $Dist_{ep,ep}(I^1) > 0$ *or* $Dist_{ec,ec}(I^1) > 0$ **then**
        $LB^1 \leftarrow +\infty$;
      else
        Compute a lower bound LB^1 for the instance I^1;

      if $Dist_{ep,ep}(I^2) > 0$ *or* $Dist_{ec,ec}(I^2) > 0$ **then**
        $LB^2 \leftarrow +\infty$;
      else
        Compute a lower bound LB^2 for the instance I^2;

      if $LB^1 > UB$ *and* $LB^2 > UB$ **then**
        return (* infeasible instance *);
      else if $LB^1 > UB$ *and* $LB^2 \leq UB$ **then**
        $I = I^2$;
      else if $LB^1 \leq UB$ *and* $LB^2 > UB$ **then**
        $I = I^1$;
      $LB = \max(LB, \min(LB^1, LB^2))$;
  return LB

If an infeasibility is detected only on I^1 (resp. I^2), then we must always fix the execution of *ep* after (resp. before) *ec* and we replace the instance I by I^2 (resp. I^1). However, If an infeasibility is detected on both I^1 and I^2, then the instance I is infeasible. The process is reiterated for each production and consumption event.

More details are given in Algorithm 3.4. Of course this test is very costly, it cannot be used in a branch and bound method at any node of the tree but only at the root.

3.4 LP-Based lower bound

3.4.1 General linear programming scheme

We now present a general linear programming scheme for computing lower bounds for ERCPSP. It is based on a decomposition of the time horizon into L successive intervals $[t_0, t_1[$, $[t_1, t_2[$, ..., $[t_{L-1}, t_L[$, which are fixed. In this formulation, ERCPSP is relaxed by allowing events to be partially executed in different intervals. The decision variable $x_{i,l}$ ($i \in X, l \in [0...L-1]$) denotes the execution proportion of event i in interval $[t_l, t_{l+1}[$, where $x_{i,l} \in [0,1]$. Thus, $a_i^k \times x_{i,l}$ represents the quantity of resource k produced or consumed in interval $[t_l, t_{l+1}[$ by event i. The number of decision variables and the number of constraints in this formulation increase proportionally with L. Let us now define formally the constraints of our linear programming formulation.

Definition 3.1. *$GLP(I)$ is the linear programming formulation built from an instance I of ERCPSP defined by:*

$$\sum_{l=0}^{L-1} x_{i,l} = 1 \qquad \forall i \in X \tag{3.1}$$

$$\sum_{l=0}^{h} \sum_{i=0}^{n+1} a_i^k \times x_{i,l} \geq 0 \qquad \forall h \in [0...L-1], \forall k \in K \tag{3.2}$$

$$\sum_{l=0}^{h_1} x_{i,l} - \sum_{l=0}^{h_2} x_{j,l} \geq 0 \qquad \forall (i,j) \in U, \forall (h_1,h_2) \in [0...L-1]^2,\ t_{h_2+1} - v_{ij} - 1 \in [t_{h_1}, t_{h_1+1}[\tag{3.3}$$

$$x_{i,l} \in [0,1] \qquad \forall l \in [0...L-1], \forall i \in X \tag{3.4}$$

$$t_0 = 0 < t_1 < ... < t_L \tag{3.5}$$

Proposition 3.3. *Given an instance I of ERCPSP, if $GLP(I)$ has no feasible solution, then the time horizon t_L of $GLP(I)$ is a valid lower bound on the optimal makespan of the instance I.*

Proof. Let I be an instance of ERCPSP. In $GLP(I)$, events can be partially executed in different intervals. Due to constraint (3.1), all events are completely executed and due to constraint (3.2), the availability of each resource $k \in K$ in each interval

$[t_l, t_{l+1}[$ is non negative. Constraint (3.3) expresses the precedence constraints. Let (i, j) be a pair of events belonging to U and let $[t_{h_1}, t_{h_1+1}[$ and $[t_{h_2}, t_{h_2+1}[$ be two intervals such that $t_{h_2+1} - v_{ij} - 1 \in [t_{h_1}, t_{h_1+1}[$ (h_1 depends on h_2). Let us consider a standard feasible schedule S of I. According to S, events i and j are executed respectively at times S_i and S_j such that $S_i + v_{ij} \leq S_j$. So, the proportion of event i processed at time S_i is larger than or equal to the proportion of event j porcessed at time $S_i + v_{ij}$. To prove the validity of constraint (3.3), we distinguish three cases:

- If $S_j \in [0, t_{h_2}[$, then there exist $h'_1 \leq h_1$ and $h'_2 < h_2$ such that $S_j \in [t_{h'_2}, t_{h'_2+1}[$ and $S_i \in [0, t_{h'_1+1}[$. This implies $\sum_{l=0}^{h_2} x_{j,l} = \sum_{l=0}^{h'_2} x_{j,l} = 1$ and $\sum_{l=0}^{h_1} x_{i,l} = \sum_{l=0}^{h'_1} x_{i,l} = 1$. So, constraint (3.3) is satisfied.

- If $S_j \in [t_{h_2}, t_{h_2+1}[$, then $x_{h_2,j} = 1$ and $S_i \in [0, t_{h_1+1}[$ (i.e., $\sum_{l=0}^{h_1} x_{i,l} = 1$). So, constraint (3.3) is also satisfied.

- If $S_j \in [t_{h_2+1}, t_L[$, then $\sum_{l=0}^{h_2} x_{j,l} = 0$ and $\sum_{l=0}^{h_1} x_{i,l} \geq 0$. So, constraint (3.3) is also satisfied.

The other constraints are valid. □ □

Note that in this formulation, the precedence and resource constraints are simultaneously taken into account and t_l is constant ($\forall l \in [0...L]$).

3.4.2 Application of the general scheme

The first lower bound we present in this section is a destructive one. It can be used to prove that no feasible solution with a makespan smaller than or equal to a value C_{max} exists. It is computed for an instance I of ERCPSP as follows. We start by fixing a trial value C_{max}. Then, we compute for each event i, its earliest starting time ES_i and its latest starting time LS_i according to C_{max}. The next step is the computation of the time-intervals. Let $T = \{t_0, t_1, ..., t_{L-1}\} = \{ES_i, LS_i, \forall i \in X\}$ and $t_L = C_{max} + 1$. We suppose without loss of generality that T is sorted in an increasing order and that all time points are different. To decrease the number of decision variables, we add to $GLP(I)$, the following linear constraints:

$$x_{i,l} = 0 \qquad \forall l \in [0...L-1], \forall i \in X,\ t_l > LS_i \tag{3.6}$$

$$x_{i,l} = 0 \qquad \forall l \in [0...L-1], \forall i \in X,\ t_{l+1} - 1 < ES_i \tag{3.7}$$

If $GLP(I)$ has no feasible solution, then $C_{max} + 1$ is a valid lower bound for the instance I. Note that with this decomposition method we obtain at most $O(n)$ time-intervals. This leads to a polynomial number of variables and of constraints.

In fact, with this decomposition method, $GLP(I)$ involves at most $O(n^2)$ variables and at most $O(n^3)$ constraints.

This lower bound can be improved by changing adequately t_l and by increasing the number of time-intervals L. In fact, given an instance I of ERCPSP and a trial value C_{max}, a stronger destructive bound can be computed using the following decomposition of the time horizon. We set $t_0 = 0$, $t_1 = t_0 + 1$,..., $t_L = t_{L-1} + 1$ and $L = C_{max} + 1$. If $GLP(I)$ has no feasible solution, then $C_{max} + 1$ is a valid lower bound for the instance I. Note that here we get a pseudo-polynomial number of time-intervals $(C_{max} + 1)$, which leads to a pseudo-polynomial number of variables and of constraints.

3.5 Computational results

Until now, no benchmark has been proposed for the ERCPSP. The only benchmark which is the most appropriate to our problem is the one proposed by [Neumann and Schwindt, 2002] for the Project Scheduling Problem with Inventory Constraints. This benchmark was considered in order to evaluate the performance of the proposed lower bounds. It consists of 360 projects with 10, 20, 50, and 100 events (NS10, NS20, NS50 and NS100) involving 5 resources, positive and negative time lags and minimization of makespan. These instances were generated according to three parameters: *network complexity* (**NC**), *resource factor* (**RF**) and *resource strength* (**RS**). Some details about this benchmark extracted from [Neumann and Schwindt, 2002][Laborie, 2002] are given in Table 3.4 such as the number of resources K, the number of instances Nb_{inst}, the number of feasible instances Nb_{feas} and the number of infeasible instances Nb_{inf}.

We also carried out experiments on 480 instances of RCPSP with 30 activities generated by authors of [Kolisch and Sprecher, 1997] (J30). All these experiments were conducted on a personal computer Intel(R) Core(TM) i7-3740QM processor with 2.70 GHz clock running GNU/Linux and all the bounds were coded in C++ language.

Instances	K	Nb_{inst}	Nb_{feas}	Nb_{inf}
NS10	5	90	60	30
NS20	5	90	43	47
NS50	5	90	48	42
NS100	5	90	47	43

Tableau 3.4 – Neumann and Schwindt benchmark details

Note that an instance of the Project Scheduling Problem with Inventory Constraints (resp. of RCPSP) must be associated with an instance of ERCPSP (as shown in Section 2.2.2) before we test our bounds. The first set of experiments was conducted to assess the performance of the following lower bounds:

- IJPPS: destructive lower bound based on JPPS,
- ER: destructive lower bound based on the energetic reasoning,
- SHIFT: shifting algorithm based bound,
- FLOW: destructive lower bound based on the flow,
- IFLOW: lower bound computed using the method described in Section 3.3.6 to improve FLOW.
- Best: equal to $\max\{IJPPS, ER, SHIFT, FLOW, IFLOW\}$,
- LB_0: critical path based lower bound.
- JPPS/RCPSP: lower bound for RCPSP based on JPPS applied to cumulative scheduling subproblems directly relaxed from RCPSP [Carlier and Néron, 2003].

Table 3.5 displays a summary of the computational results that were obtained on the instances of Neumann and Schwindt [Neumann and Schwindt, 2002] and Table 3.6 displays the computational results that were obtained on J30. For each lower bound, we provide: $\%Gap$: the average deviation from the optimal makespan in percent, $\%Opt$: the percentage of optimal makespans found and rt_{avg}: the mean running time in seconds.

A first observation from these results is that all the proposed lower bounds outperform the classical lower bound LB_0. They are very efficient on the instances of [Neumann and Schwindt, 2002], but less efficient on the instances of RCPSP [Kolisch and Sprecher, 1997]. More precisely, we observe that IFLOW exhibits the best performance on the instances of Neumann and Schwindt. In fact, it yields the tightest average deviation $\%Gap$ and the largest percentage of optimal makespans reached $\%Opt$. However, ER is the best bound on J30 and IJPPS is slightly better than JPPS/RCPSP which proves the efficiency of our extraction method. Moreover, we see that:

- On NS10 and NS20, ER and FLOW yield the second best average deviation $\%Gap$. IJPPS and SHIFT provide also good bounds and are impressively fast.
- On NS50, FLOW and SHIFT exhibit the second best results after IFLOW.

		$\%Gap$	$\%Opt$	rt_{avg}
NS10	IJPPS	0.405%	93.333%	0.001
	ER	0.239%	**98.333%**	0.023
	SHIFT	0.502%	93.333%	**0.000**
	FLOW	0.239%	**98.333%**	0.001
	IFLOW	**0.077%**	**98.333%**	0,025
	LB_0	2.888%	81.666%	**0.000**
	BEST	**0.077%**	**98.333%**	-
NS20	IJPPS	2.342%	81.395%	0.005
	ER	1.938%	83.721%	0.119
	SHIFT	2.133%	83.721%	0.003
	FLOW	1.938%	83.721%	0.008
	IFLOW	**1.526%**	**86.046%**	0,311
	LB_0	8.858%	51.162%	**0.002**
	BEST	**1.526%**	**86.046%**	-
NS50	IJPPS	1.572%	79.167%	0.040
	ER	1.572%	79.167%	1.543
	SHIFT	1.538%	81.250%	0.034
	FLOW	1.522%	81.250%	0.077
	IFLOW	**0.713%**	**83.333%**	32.00
	LB_0	14.501%	34.501%	**0,033**
	BEST	**0.713%**	**83.333%**	-
NS100	IJPPS	0.764%	**97.872%**	0.263
	ER	0.764%	**97.872%**	11.24
	SHIFT	0.764%	**97.872%**	0.248
	FLOW	0.764%	**97.872%**	0.638
	IFLOW	**0.648%**	**97.872%**	> 600
	LB_0	18.012%	26.000%	**0,245**
	BEST	**0.648%**	**97.872%**	-

Tableau 3.5 – Results of the bounds on the benchmark of Neumann and Schwindt

	$\%Gap$	$\%Opt$	rt_{avg}
IJPPS	7.148%	46.208%	0,059
ER	**5.476%**	**53.125%**	2.901
SHIFT	8.340%	46.000%	0,055
FLOW	8.128%	48.333%	0,060
IFLOW	7.389%	51.667%	39.900
LB_0	10.020%	44.000%	**0.001**
BEST	**5.385%**	**54.791%**	-
JPPS/RCPSP	7.250%	46.208%	0.002

Tableau 3.6 – Results of the bounds on J30

Instances	$FLOW$	$IFLOW$
NS10	40.00%	96.66%
NS20	55.31%	87.23%
NS50	40.47%	85.41%
NS100	54.54%	54.54%

Tableau 3.7 – Percentage of infeasible instances found

- On NS100, IJPPS, ER, SHIFT and FLOW have a similar overall performance. Even though, IJPPS and SHIFT are the quickest.
- On J30, IJPPS and the energetic reasoning are better than the other bounds. The energetic reasoning is more powerful than IJPPS but is too costly in computations.

Thus, we can deduce that our lower bounds are not comparable.

Another striking observation is that IFLOW is very efficient for small and medium instances while it is not for large instances. In fact, IFLOW improves significantly FLOW on NS10, NS20 and NS50. However, it is too costly in computation time and its improvement is not much significant on NS100.

In addition to their excellent performances on the Neumann and Schwindt benchmark, FLOW and IFLOW were able to detect some infeasible instances contrary to the other lower bounds. Table 3.7 summarizes the percentage of infeasible instances detected using these two bounds. As we can observe, IFLOW improves dramatically the percentage of detections on the small and medium instances (NS10, NS20 and NS50). However, it exhibits the same results as the basic bound FLOW on the large instances (NS100).

Note that we have also tested the method described in Section 3.3.6 with the other bounds but their performances were not improved.

To evaluate the performance of our two destructive lower bounds, we also considered the benchmark proposed by (Neumann and Schwindt 2003). The obtained

results are very interesting (see Table 3.8). They are very close to the optimal makespan. In fact, more than 93% instances are closed with an average deviation in percent smaller than 0.8%, which is better than the other proposed lower bounds, where only 90% instances were closed with 1.12% average deviation in percent. The second lower bound with pseudo-polynomial decomposition (GLP2) yields the tightest average deviation (0.6%), but it is too costly in computation time. Indeed, its average computation time on the large instances is equal to 4 minutes, while the average computation time of the first lower bound (GLP1) is less than 15 seconds.

Data	GPL1			GPL2		
	Gap	Opt	CPU	Gap	Opt	CPU
NS10	0.0	100.0	0.0	0.0	100.0	0.0
NS20	1.2	88.7	1.0	1.0	91.0	3.0
NS50	1.4	87.0	5.0	1.2	88.0	40.0
NS100	0.7	97.8	15.0	0.6	97.8	$<$240

Tableau 3.8 – Results of the LP-based lower bounds

3.6 Conclusion

In this paper, we have studied the Extended Resource Constrained Project Scheduling Problem (ERCPSP) which is an extension of the RCPSP. We have shown the usefulness of this model by presenting how several classical project scheduling problems can be modeled by it. Moreover, we have proposed six new lower bounds for this problem. Two of them are based on the extraction of a generalized Cumulative Scheduling Problem, combined with an adapted version of Jackson's pseudo-preemptive scheduling scheme [Carlier and Pinson, 2004] and the concept of energetic reasoning. Two further lower bounds respectively result from applying Carlier and Rinnooy Kan's Shifting Algorithm to a financing problem relaxation and iteratively testing the feasibility of appropriate Network Flow Problems in a dichotomic search method. The last two lower bounds are destructive lower bounds computed using a general linear programming scheme.

We have provided a new mechanism translating a scheduling problem with depleting and replenishing events into a tight relaxation that exclusively contains renewable resources. Since many lower bounding techniques are available for the case of renewable resource constraints, this approach paves the way for deriving further bounding procedures. We have also proposed a method to improve these lower bounds by adding new precedence constraints.

We observe that our lower bounds dramatically improve the critical path based lower bound LB_0 and are very close to the optimal makespans for Neumann and Schwindt instances, we also observed that they are not directly comparable. As perspectives, we aim to build a branch-and-bound method to solve the ERCPSP using these lower bounds.

Exact Procedures for the ERCPSP

Sommaire

4.1 Introduction 79
4.2 MILP formulations for the ERCPSP 80
4.3 Branch-and-Bound method for ERCPCP 88
4.4 Conclusion 93

4.1 Introduction

The Extended Resource Constrained Project Scheduling Problem (ERCPSP) is defined by events, non-renewable resources and generalized precedence constraints between pairs of events ([Carlier et al., 2009]). Each event has the ability to produce or to consume some units of resources at its occurrence time. The objective is to build a schedule that satisfies precedence and resource constraints and minimizes the makespan.

ERCPSP is an extension of the Resource Constrained Project Scheduling Problem (RCPSP), where activities requiring renewable resources are replaced by events consuming or producing non-renewable resources. There exists in the literature a large number of MILP formulations for the RCPSP. Among others, we can quote the formulations with an exponential number of variables such as the discrete time formulation of [Mingozzi et al., 1998], and the formulation of [Moukrim et al., 2015]. On the other hand, there exist some formulations containing a polynomial number of variables and other formulations containing a pseudo-polynomial number of variables. In this paper, we restrict our study to these two latter categories because the objective is to propose formulations that allow solving problems by using directly a MILP solver.

In this work, we have been inspired by previous works on RCPSP and RCPSP/CPR (see [Artigues et al., 2003], [Koné et al., 2013]) to afford four mixed

integer linear programming formulations to solve the ERCPSP. More precisely, we propose first an adaptation of two known (time-indexed) MILP formulations of RCPSP to ERCPSP. Second, we introduce an adaptation of a flow-based continuous-time formulation. Finally, we propose a new MILP formulation based on the concept of event partitioning to solve the problem.

The remaining of this chapter is organized as follows. In Section 4.2, we present our four MILP formulations for ERCPSP and we report experimental results. In Section 4.3, we present a branch-and-bound method to solve the ERCPSP using our lower bounds, and we adapt two constraint propagation algorithms to improve this method. Finally, we conclude the paper in Section 4.4.

4.2 MILP formulations for the ERCPSP

In this section, we propose first an adaptation of two known time-indexed formulations of RCPSP to ERCPSP. Second, we introduce an adaptation of a flow-based continuous-time formulation. Finally, we propose a MILP formulation based on the concept of event group for ERCPSP.

4.2.1 Discrete-time formulation (DT)

The discrete-time formulation (DT) was initially introduced for the RCPSP by [Pritsker et al., 1969]. Then, it was adapted by [Koné et al., 2013] to the RCPSP with Consumption and Production of Resources. In this subsection, we extend this formulation to the ERCPSP. The DT formulation involves only one type of binary decision variable, x_{it}, indexed by both events and time. The decision variable is defined so that $x_{it} = 1$ if event i occurs at time t, and $x_{it} = 0$ otherwise. So, this formulation can be written for ERCPSP as follows:

$$\min \sum_{t=ES_{n+1}}^{LS_{n+1}} t x_{n+1,t} \tag{4.1}$$

$$\sum_{t=ES_i}^{LS_i} t x_{it} + v_{ij} \leq \sum_{t=ES_j}^{LS_j} t x_{jt} \qquad \forall (i,j) \in U \tag{4.2}$$

$$\sum_{\tau=0}^{t} \sum_{i=0}^{n+1} a_i^k x_{i\tau} \geq 0 \qquad \forall k \in K, \forall t \in H \tag{4.3}$$

$$\sum_{t=ES_i}^{t=LS_i} x_{it} = 1 \qquad \forall i \in X \tag{4.4}$$

$$x_{it} = 0 \qquad \forall i \in X, \forall t \in H \setminus \{ES_i, ..., LS_i\} \tag{4.5}$$

$$x_{it} \in \{0, 1\} \qquad \forall i \in X, \forall t \in \{ES_i, ..., LS_i\} \tag{4.6}$$

The objective function is given by (4.1). For each event $i \in X$, the value of S_i can be recovered through the relation: $S_i = \sum_{t=ES_i}^{LS_i} t x_{it}$. Constraints (4.2) are simple translations of precedence constraints. Constraints (4.3) simply express the resource constraints. They ensure to have a non negative resource availability for each resource $k \in K$ at any time t. Note that these constraints take into account the production and consumption of resources. Constraints (4.4) and (4.6) impose that each event is processed exactly one time over the planning horizon T. Constraints (4.5) mean that each event is executed between its earliest occurrence time and its latest occurrence time.

Remark that this formulation involves $\sum_{i=0}^{n+1}(LS_i - ES_i)$ binary variables, and $|U| + (T+1)|K| + n + 1$ constraints.

4.2.2 Disaggregated discrete-time formulation (DDT)

The disaggregated discrete-time formulation (DDT) was proposed for the RCPSP by [Christofides et al., 1987]. It was adapted by [Koné et al., 2013] to the RCPSP/CPR. Here, we extend this formulation to the ERCPSP. The DDT formulation is similar to the DT formulation. The unique difference between them is in the formulation of the precedence constraints. In fact, the DT formulation involves one constraint for each precedence relation, while the DDT formulation defines one constraint for each precedence relation and for every time of the scheduling horizon:

$$\sum_{t=\tau}^{LS_i} x_{it} + \sum_{t=ES_j}^{\min\{LS_j, \tau+v_{ij}-1\}} x_{jt} \leq 1 \quad \forall (i,j) \in U, \forall \tau \in [ES_i, LS_i] \tag{4.7}$$

All the other constraints remain the same. Note that constraints (4.7) and (4.4) imply constraints (4.2). The DDT and DT require the same number of binary variables. However, DDT has $|U| \times (LS_i - ES_i)$ more constraints.

In time-indexed formulations, the number of binary variables increases proportionally with the time horizon T. We remark that both DDT and DT require pseudo-polynomial numbers of constraints and variables. As a consequence, they exhibit disastrous performances when solving problems with a very large time horizon.

4.2.3 Flow-based continuous-time formulation

[Artigues et al., 2003] proposed for the RCPSP a compact flow-based continuous-time (FCT) formulation. This formulation was extended to the RCPSP/CPR by [Koné et al., 2013]. Inspired by these two works, we propose here a flow-based continuous-time formulation for the ERCPSP. The idea is as follows. Each quantity of resources produced by a production event is transfered to consumption events that follow according to precedence constraints.

The FCT formulation involves three types of decision variables. First, a sequential binary variable y_{ij} is needed for each pair of events (i, j) to determine whether event j is processed after event i. Second, a continuous time variable S_i is required for each event i to determine its occurrence time. Finally, for each resource $k \in K$ and for each pair of events $(i, j) \in X_k^p \times X_k^c$, a continuous flow variable f_{ijk} is introduced to indicate the quantity of resource k that is transferred from event i (at its occurrence time) to event j.

The FCT formulation can be written as follows:

$$\min S_{n+1} \tag{4.8}$$

$$1 \leq y_{ij} + y_{ji} \leq 2 \qquad \forall (i,j) \in X^2, i < j \tag{4.9}$$

$$y_{il} + 1 \geq y_{ij} + y_{jl} \qquad \forall (i,j,l) \in X^3 \tag{4.10}$$

$$S_j - S_i \geq v_{ij} \qquad \forall (i,j) \in U \tag{4.11}$$

$$S_j - S_i \geq M_{ij}(y_{ij} - 1) \qquad \forall (i,j) \in X^2 \tag{4.12}$$

$$f_{ijk} \leq \min(a_i^k, |a_j^k|) y_{ij} \qquad \forall k \in K, \forall (i,j) \in X_k^p \times X_k^c \tag{4.13}$$

$$\sum_{i \in X_k^p} f_{ijk} = |a_j^k| \qquad \forall k \in K, \forall j \in X_k^c \tag{4.14}$$

$$\sum_{j \in X_k^c} f_{ijk} \leq a_i^k \qquad \forall k \in K, \forall i \in X_k^p \tag{4.15}$$

$$ES_i \leq S_i \leq LS_i \qquad \forall i \in X \tag{4.16}$$

$$y_{ij} = 1 \qquad \forall (i,j) \in U, v_{ij} \geq 0 \tag{4.17}$$

$$y_{ji} = 0 \qquad \forall (i,j) \in U, v_{ij} > 0 \tag{4.18}$$

$$f_{ijk} \geq 0 \qquad \forall k \in K, \forall (i,j) \in X_k^p \times X_k^c \tag{4.19}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in X^2 \tag{4.20}$$

where M_{ij} is an upper bound for $S_i - S_j$, which can be fixed to $ES_i - LS_j$. Constraints (4.9) mean that for two distinct events i and j, either i precedes j ($y_{ij} = 1, y_{ji} = 0$), or j precedes i ($y_{ij} = 0, y_{ji} = 1$), or i and j are executed at the same time ($y_{ij} = 1, y_{ji} = 1$). Constraints (4.10) define the transitivity of the precedence relations. Constraints (4.11) express the precedence constraints. Constraints (4.12) link the time variable of event i and event j with the sequential binary variable y_{ij} for each $(i,j) \in X^2$. If $y_{ij} = 1$ (i precedes j), the constraint enforces the precedence relation $S_i \leq S_j$, whereas if $y_{ij} = 0$, the constraint is always satisfied. Constraints (4.13) link the sequential binary variables with the flow variables. The maximum flow sent from i to j is limited to the capacity $\min(a_i^k, |a_j^k|)$ if i precedes j, and to 0 otherwise for each $k \in K$ and $(i,j) \in X_k^p \times X_k^c$. Constraints (4.14) and (4.15) are the usual inequalities of flow conservation. Constraints (4.16) restrain the occurrence time of any event $i \in X$ to lie between its earliest occurrence time ES_i and its latest occurrence time LS_i. Constraints (4.17) and (4.18) fix the preexisting precedence constraints.

It was shown by [Applegate and Cook, 1991] that this formulation produces bad linear-relaxation bounds due to big-M constants in constraints (4.12). However, it may be preferable to time-indexed formulations to solve problems with a large time horizon. In fact, DT and DDT both involve a pseudo-polynomial number of variables and constraints, while FCT has a polynomial number of variables and constraints. It involves at most $O(|K| \times n^2)$ decision variables and at most $O(n^3)$ constraints.

4.2.4 Event partitioning based formulation (EP)

In contrast to the formulations involving variables indexed by time, we propose here a new formulation for the ERCPSP using variables indexed by event subsets, that we call the event partitioning based formulation (EP). The idea is to construct feasible schedules by partitioning the set of events into several subsets such that each subset contains events having the same occurrence time. The number of subsets can be restricted to the number of events.

Let $\Phi = \{\phi_0, \phi_1, ..., \phi_{n+1}\}$ be a partition of X. The event partitioning based formulation uses only one type of binary variables. A decision variable z_{ie} is equal to 1 if event i belongs to subset ϕ_e, and to 0 otherwise. This formulation requires

also the introduction of the following continuous variables:

- S_i: the occurrence time of event i.
- t_e: the occurrence time of each event included in ϕ_e.
- s_{ek}: the availability of resource k after the execution of each event of ϕ_e.

The EP formulation can be written as follows:

$$\min t_{n+1} \tag{4.21}$$

$$t_0 = 0 \tag{4.22}$$

$$t_e \geq S_i - M(1 - z_{ie}) \qquad \forall i \in X, \forall e \in [0...n+1] \tag{4.23}$$

$$S_i \geq t_e - M(1 - z_{ie}) \qquad \forall i \in X, \forall e \in [0...n+1] \tag{4.24}$$

$$t_{e+1} \geq t_e \qquad \forall e \in [0...n] \tag{4.25}$$

$$\sum_{e=0}^{n+1} z_{ie} = 1 \qquad \forall i \in X \tag{4.26}$$

$$S_i + v_{ij} \leq S_j \qquad \forall (i,j) \in U \tag{4.27}$$

$$s_{0k} = \sum_{i=0}^{n+1} a_i^k z_{i0} \qquad \forall k \in K \tag{4.28}$$

$$s_{ek} = s_{e-1,k} + \sum_{i=1}^{n+1} a_i^k z_{ie} \qquad \forall k \in K, \forall e \in [1...n+1] \tag{4.29}$$

$$s_{ek} \geq 0 \qquad \forall e \in [0...n+1], k \in K \tag{4.30}$$

$$ES_i \leq S_i \leq LS_i \qquad \forall i \in X \tag{4.31}$$

$$t_e \geq ES_i z_{ie} \qquad \forall i \in X, \forall e \in [0...n+1] \tag{4.32}$$

$$t_e \leq LS_i z_{ie} + LS_{n+1}(1 - z_{ie}) \qquad \forall i \in X, \forall e \in [0...n+1] \tag{4.33}$$

$$ES_{n+1} \leq t_{n+1} \leq LS_{n+1} \tag{4.34}$$

$$t_e \geq 0 \qquad \forall e \in [0...n+1] \tag{4.35}$$

$$z_{ie} \in \{0,1\} \qquad \forall i \in X, \forall e \in [0...n+1] \tag{4.36}$$

where M is a large enough constant, which can be set to any upper bound on the makespan of the ERCPSP instance, we can set M for example to LS_{n+1}.

The objective function is given by (4.21). Constraint (4.22) stipulates that each event belonging to ϕ_0 is processed at time 0. Constraints (4.23) and (4.24) ensure that events belonging to a same subset have the same occurrence time. Constraints (4.25) order the subsets according to the occurrence time of their events. Constraints (4.26) require that each event has a single occurrence. Constraints (4.27) express the precedence constraints. Constraints (4.28) give the availability of each resource at time 0. Constraints (4.29) give the level of each resource after the execution of each subset events. Constraints (4.30) express the resource constraints, they ensure non negativity for each resource level. Constraints (4.31)-(4.34) are valid inequalities based on event time windows. They state that the execution of each event $i \in X$ must lie between its earliest occurrence time ES_i and its latest occurrence time LS_i. occurrence Note that the variables s_{ek} can all be substituted by their expression in function of z_{ie}. Thus, constraints (4.28)-(4.30) can be replaced by:

$$\sum_{e'=0}^{e} \sum_{i=0}^{n+1} a_i^k z_{ie'} \geq 0 \qquad \forall k \in K, \forall \phi_e \in \Phi \tag{4.37}$$

The EP formulation involves at most $O(n^2)$ variables and at most $O(n^2)$ constraints. Compared with DT and DDT, this formulation contains a polynomial number of variables and constraints, however, it involves big-M constraints. Compared with FCT, EP has less variables and less constraints.

4.2.5 Computation results

To evaluate the performance of our four MILP formulations, we considered the benchmark proposed by [Neumann and Schwindt, 2002] for the Project Scheduling Problem with Inventory Constraints. It consists of 360 projects with 10, 20, 50, and 100 events, involving 5 resources, min/max delays between events and minimization of makespan.

We performed a series of tests to compare the lower bounds obtained by linear relaxation of each of the four formulations DT, DDT, FCT and EP. These tests were carried out on a personal computer Intel(R) Core(TM) i7-3740QM processor with 2.70 GHz clock running GNU/Linux. The formulations were coded in C++ language and the solver used was ILOG-CPLEX (version 12.6). We limited the computation time of each instance to 300s.

Note that an instance of the Project Scheduling Problem with Inventory

Constraints must be associated with an instance of ERCPSP before the linear relaxation based lower bounds are computed.

Tableau 4.1 – Linear relaxation results.

Instances	Bounds	$\%Gap$	$\%Opt$	$Time\ (s)$
NS10	DT	0.4	95.0	0.0
	DDT	0.2	98.3	0.0
	FCT	0.4	96.6	0.0
	EP	0.5	93.3	0.0
	LB_{best}	0.2	98.3	0.0
NS20	DT	2.2	81.4	0.2
	DDT	1.8	83.7	2.1
	FCT	2.4	81.4	0.0
	EP	2.4	81.4	0.0
	LB_{best}	1.9	83.7	0.1
NS50	DT	1.5	79.1	2.7
	DDT	1.1	81.2	10.2
	FCT	1.5	79.1	1.6
	EP	1.5	79.1	0.4
	LB_{best}	1.5	81.2	1.5
NS100	DT	0.7	97.8	19.0
	DDT	0.6	97.8	40.0
	FCT	0.7	97.8	80.0
	EP	0.7	97.8	6.0
	LB_{best}	0.7	97.8	2.0

Table 4.1 displays a summary of the computational results that are obtained on the instances of [Neumann and Schwindt, 2002]. In this table, LB_{best} represents the best lower bound obtained in ([Sahli et al., 2015]). We also use the following abbreviations:

- $\%Gap$ gives the average deviation in percent for solved instances from the optimal makespans.

- $\%Opt$ provides the percentage of optimal makespans found.

- $Time(s)$ displays the average CPU time required, in seconds.

The methods are ranked according to criteria $\%Gap$, $\%Opt$, $Time$ (s), by order of importance.

A first observation from these results is that all the linear relaxation based lower bounds are efficient on the instances of [Neumann and Schwindt, 2002] and are computed within 80s. In fact, they are very close to the optimal makespans.

More precisely, we observe that DDT produces the best lower bounds. It yields the tightest average deviation $\%Gap$ and the largest percentage of optimal makespans reached $\%Opt$. It also slightly improved the best lower bound (LB_{best}) proposed in ([Sahli et al., 2015]). LB_{best} is slightly better than the other LP-based lower bounds. Moreover, we see that:

- On NS10 and NS20, the linear relaxations of DT and FCT present the second best average deviation $\%Gap$. EP yields also good bounds and is impressively fast.

- On NS50, the LP-relaxations of DT, FCT and EP exhibit a similar overall performance. Even though, EP is the fastest, followed by FCT.

- On NS100, the lower bound given by DT, FCT and EP are similar and EP remains the fastest. But here DT is faster than FCT.

The LP-relaxation of the time-indexed formulation is better than the other ones. However, the number of variables of this formulation explodes for instances with a large time horizon. In fact, we observe that the LP size increases dramatically for instances involving very large scheduling horizon.

Another striking observation is that even if DT and DDT are very similar and use the same decision variables, the linear relaxation of DDT is tighter than the one of DT. This result is consistent with previous studies of lower bounds using discrete time relaxations for the RCPSP ([Koné et al., 2011]).

In terms of exact (integer) solving, the four MILP formulations are able to solve the instances involving 10 and 20 events in less than 120s. But they cannot solve the large instances within 300s.

4.3 Branch-and-Bound method for ERCPCP

The search tree which we propose is a binary tree, each node represents a subset of solutions which satisfy a set of precedence constraints α. At each branching phase, from a given node, we partition the current subset of solutions into two disjoint subsets $\alpha \cup \{(a,b)\}$ where $v_{ab} = 0$ and $\alpha \cup \{(b,a)\}$ where $v_{ba} = 1$. Two events a and b are chosen as follows. From the current subset which contains arcs set α, we calculate the earliest starting time of each event in respecting graph $G = (X, U \cup \alpha)$. Then we draw the curve of level of resources. For the first time t when we encounter a conflict of resource $k \in K$, we pick all events whose earliest starting times are earlier than t and who consume resource k. We calculate their criticality $Crit = a_{bk}/(t - t_b)$ and we choose the one with max criticality as event b. For the selection of event a, it's nearly the same. We pick all events whose earliest starting times are not earlier than t and who produce resource k, we calculate their criticality $Crit = a_{ak}/(t_a - t)$ and we choose the one with max criticality to be event a. If there is no resource conflict, that means we have obtained a feasible solution.

A naive upper bound ub of GRCPSP can be calculated as introduced in [Carlier et al., 2009]:

$$ub = \sum_{j=1}^{n} \max\{\max\{0, v_{ji}\} | \forall i, (j,i) \in U\} \tag{4.38}$$

Let $EST(\alpha)$ denote the earliest start time solution for graph $G = (X, U \cup \alpha)$. The enumeration is performed according to a depth-first search strategy. We initialize the upper bound ub according to equation 4.38. We start with the root which contains the subset $\alpha_0 = \emptyset$, if $EST(\alpha_0)$ is feasible, we set ub equal to $C(EST(\alpha_0))$ and backtrack. Otherwise, we define two child nodes p_{left} and p_{right} as described before. For each child node p, if $lb_p < ub$, we add it into the search tree. We branch from one of the child nodes with maximum lower bound. The branch-and-bound procedure terminates when all nodes in the processing tree are exploited. Note that to evaluate each node we use the lower bounds presented in the previous chapter.

4.3.1 Constraint propagation

In Constraint Programming, a partial schedule is a set of decision variables (occurrence time, requirement of resource) and a set of constraints between these variables (temporal and resource constraints). An instantiation of all these decision variables that satisfies all the constraints represents a solution. Constraint propagation is

the main technique used in Constraint Programming to reduce the search space. It consists in removing from the possible values of a decision variable the ones that surely violate some constraint. This method can be used in a branch-and-bound procedure to find features shared by all the solutions reachable from the current search node. These features may involve some additional constraints that must be satisfied or some domain restriction. In this subsection, we present two adapted constraint propagation algorithms to improve our branch-and-bound method: the timetabling of [Le Pape, 1994] and the balance constraint of [Laborie, 2002].

4.3.1.1 Timetabling

Timetabling is based on the computation for each time t the minimal resource usage [Laborie, 2002]. It permits to reduce the domains of the occurrence times of events by removing the dates that would lead to an over-consumption of the resource. The main advantage of the timetabling technique is its low algorithmic complexity which is linear [Laborie, 2002]. For this reason, it is the main technique used today for large scheduling problems.

Suppose that $ES(i)$ and $LS(i)$ are respectively the earliest and the latest occurrence times of event i. Thus we know surely that i will be executed before $LS(i)$. For each resource k, the minimal resource usage of event i at time t is given by the function:

$$U_k(i,t) = \begin{cases} a_i^k & \text{if } t > LS(i) \text{ or } ES(i) \leq t \leq LS(i) \text{ and } a_i^k \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Note that if i produces a quantity a_i^k of resource k and $t \geq ES(i)$, then the minimal resource usage of i is equal to a_i^k. A curve $U_k(t)$ is maintained which aggregates all these demands:

$$U_k(t) = \sum_{i \in X} U_k(i,t).$$

If there exists a time t such that $U_k(t)$ is negative, then the current schedule cannot lead to a solution and the search must backtrack. Furthermore, if there exists a production event p and a time t such that $U_k(t) - U_k(p,t) < 0$, then event p must occur before t. It would be otherwise an over-consumption of the resource. Moreover, if there exists a consumption event c and a time t such that $U_k(t) - U_k(p,t) + a_c^k < 0$, then event c must occur after t. It would be otherwise an over-consumption of the resource. Thus timetabling allows us to increase (resp. decrease) the earliest start times (resp. latest completion times) of events.

4.3.1.2 Balance constraint

The balance constraint was introduced by [Laborie, 2002] for scheduling problems with reservoir resources. The idea of the balance constraint is to compute, for each event $i \in X$ and for each resource $k \in K$, a lower and an upper bound on the resource level just before $(S(i) - \varepsilon)$ and just after $(S(i) + \varepsilon)$ event i. The balance constraint can be adapted to ERCPSP by computing only upper bounds on the resource level. Given an event $i \in X$, let us define the following subsets of events:

- $X^s(i) = \{j \in X \mid l_{i,j} = l_{j,i} = 0\}$ is the set of events simultaneous with i.
- $X^b(i) = \{j \in X \mid l_{i,j} < 0 \text{ and } l_{j,i} > 0\}$ is the set of events strictly before i.
- $X^{bs}(i) = \{j \in X \mid l_{i,j} < 0 \text{ and } l_{j,i} = 0\}$ is the set of events before i.
- $X^a(i) = \{j \in X \mid l_{i,j} > 0 \text{ and } l_{j,i} < 0\}$ is the set of events strictly after i.
- $X^{as}(i) = \{j \in X \mid l_{i,j} = 0 \text{ and } l_{j,i} < 0\}$ is the set of events after or simultaneous with i.
- $X^u(i) = \{j \in X \mid l_{i,j} < 0 \text{ and } l_{j,i} < 0\}$ is the set of events unranked with respect to i.

For each event i and each resource k, an upper bound on the resource level $L^{<}_{max}(i,k)$ at date $S(i) - \varepsilon$ (just before i) is computed assuming:

- All the events belonging to subset $X^b(i)$.
- All the production events belonging to $X^{bs}(i) \cup X^u(i)$.

In a similar way, an upper bound on the resource level $L^{>}_{max}(i,k)$ at date $S(i) + \varepsilon$ (just after i) is computed assuming:

- All the events belonging to subset $X^b(i) \cup X^s(i) \cup X^{bs}(i)$.
- All the production events belonging to subset $X^{as}(i) \cup X^u(i)$.

More formally, if P^k is the set of production events of resource k, then these upper bounds can be computed as follows:

$$L^{<}_{max}(i,k) = \sum_{j \in X^b(i)} a^k_j + \sum_{j \in P^k \cap (X^{bs}(i) \cup X^u(i))} a^k_j \tag{4.39}$$

$$L^{>}_{max}(i,k) = \sum_{j \in X^b(i) \cup X^s(i) \cup X^{bs}(i)} a^k_j + \sum_{j \in P^k \cap (X^{as}(i) \cup X^u(i))} a^k_j \tag{4.40}$$

For each upper bound, the balance constraint for ERCPSP is able to discover tree kind of information: dead ends, new time windows for events and new precedence relations.

Discovering dead ends : If $L^{<}_{max}(i,k) < 0$ (resp. $L^{>}_{max}(i,k) < 0$), then the level of resource k will surely be negative just before (resp. after) event i. Thus, the current schedule cannot lead to a solution and the search must backtrack.

Discovering new time windows for events : Let us define $\Psi^{<}(i,k)$ and $\Psi^{>}(i,k)$ as follows:

$$\Psi^{<}(i,k) = -\sum_{j \in X^b(i)} a^k_j \tag{4.41}$$

$$\Psi^{>}(i,k) = -\sum_{j \in X^b(i) \cup X^s(i) \cup X^{bs}(i)} a^k_j \tag{4.42}$$

If $\Psi^{<}(i,k)$ is positive, it means that some production events in $X^{bs}(i) \cup X^u(i)$ must be executed strictly before i to produce at least: $\Psi^{<}(i,k)$.

Let $\{ep_1, ..., ep_s\}$ be the set of production events in $X^{bs}(i) \cup X^u(i)$. We suppose that these events are indexed in a nondecreasing order of their earliest occurrence time. Let r be the index in $[1, s]$ such that:

$$\sum_{l=1}^{r-1} a^k_{ep_l} < \Psi^{<}(i,k) \leq \sum_{l=1}^{r} a^k_{ep_l}$$

If event i is executed at a date $S(i) < ES(ep_r)$, not enough production events could be executed strictly before i to ensure a positive level of resource. Thus, $ES(ep_r)$ is a valid lower bound of $S(i)$.

In the same way, we can found a new lower bound of $S(i)$ if $\Psi^{>}(i,k)$ is positive.

Discovering new precedence relations : Let P^k be the set of production events of resource k. Suppose there exists a production event ep in $X^{bs}(i) \cup X^u(i)$ such that:

$$\sum_{j \in P^k \cap (X^{bs}(i) \cup X^u(i)) \cap (X^b(ep) \cup X^{bs}(ep) \cup X^u(ep))} a^k_l < \Psi^{<}(i,k)$$

Then, if we had $S(i) < S(ep)$, there is no way to produce $\Psi^{>}(i,k)$ before event i. In fact, the only events which can produce strictly before event i are the ones in $P^k \cap (X^{bs}(i) \cup X^u(i)) \cap (X^b(ep) \cup X^{bs}(ep) \cup X^u(ep))$. Thus, we deduce the necessary precedence constraint $S(ep) < S(i)$.

In the same way, if there exists a production event ep in $X^{as}() \cup X^{u}(i)$ such that:

$$\sum_{j \in P^k \cap (X^{as}(i) \cup X^u(i)) \cap (X^b(ep) \cup X^{bs}(ep) \cup X^u(ep)))} a_l^k < \Psi^<(i,k)$$

Then, we deduce the precedence relation $S(ep) \leq S(i)$

4.3.2 Computation results

To evaluate the performance of our branch-and-bound method, we have considered the benchmark of [Neumann and Schwindt, 2002]. This benchmark is the most appropriate to our problem. It consists of 360 projects with 10, 20, 50, and 100 events involving 5 resources, positive and negative time lags and minimization of makespan. From these 300 problems, 12 hard instances were not solved to optimality by the approach of [Neumann and Schwindt, 2002]. These hard instances were solved by the approach of [Laborie, 2002]. We tested our search procedure combined with the shifting algorithm lower bound on these 12 hard instances. All the other problems were easily solved using our approach in less than 10 seconds.

Instance	n	Optimal	Neumann and Schwindt			Laborie			Our approach		
			UB	*proof*	CPU	UB	*proof*	CPU	UB	*proof*	CPU
10	50	92	93	-	120	92	yes	0.3	92	yes	0.0
27	50	96	$+\infty$	-	120	96	yes	2.4	96	yes	10.0
82	50	inf	$+\infty$	-	120	$+\infty$	yes	0.1	$+\infty$	yes	0.1
6	100	211	223	-	120	211	yes	1.0	211	no	120
12	100	197	197	-	120	197	yes	0.7	200	-	120
20	100	199	217	-	120	199	yes	0.5	199	yes	40
30	100	204	218	-	120	204	yes	2.1	205	-	120
41	100	337	364	-	120	337	yes	0.6	337	yes	50
43	100	inf	$+\infty$	-	120	$+\infty$	yes	7.7	$+\infty$	yes	2.0
54	100	344	360	-	120	344	yes	0.5	344	yes	8.0
58	100	317	326	-	120	317	yes	0.5	317	yes	10.0
69	100	inf	$+\infty$	-	120	$+\infty$	yes	2.0	$+\infty$	yes	0.7

Tableau 4.2 – Results of the branch-and-bound method

The experiments were conducted on a personal computer Intel(R) Core(TM) i7-3740QM processor with 2.70 GHz clock running GNU/Linux and the method was coded in C++ language. Table 4.2 summarizes the results obtained by our method, the beam search of [Neumann and Schwindt, 2002] and the approach of [Laborie, 2002] on the 12 hard instances. The column n provides the number of events of each instance. The column Optimal gives the optimal makespan of each instance. We provide for each method and for each instance:

- UB: the best upper bound obtained.
- *proof*: column to show if the optimality is proved or not.

- CPU: CPU time in seconds (we limited the computation time of each instance to 2 minutes).

We can see that 9 among the 12 hard problems are closed in less than 50 seconds CPU time, which is better than the method of Neumann and Swchindt. Comparing with the approach of Laborie, our method is competitive for these 9 instances. The three remaining instances (6, 12 and 30) are hard for our procedure but they are not hard for the approach of Laborie. All the other problems were easily solved using our approach in less than 10 seconds.

4.4 Conclusion

Inspired by previous works on RCPSP, we have proposed four mixed integer linear programming models to solve this problem. The first one is an adaptation of the discrete-time formulation (DT) which was initially introduced for the RCPSP by [Pritsker et al., 1969]. The second one is an adaptation of the disaggregated discrete-time formulation (DDT) which was initially proposed by [Christofides et al., 1987]. The third one is an adaptation of a flow-based continuous-time formulation (FCT) and the last one is an event partitioning based formulation (EP). To evaluate the performance of our four MILP formulations, we have considered the benchmark proposed by [Neumann and Schwindt, 2002] for the Project Scheduling Problem with Inventory Constraints. We have compared the lower bounds obtained by linear relaxation of each of the four MILP formulations. The obtained results are very interesting. In fact, all the linear relaxation based lower bounds are very close to the optimal makespans and are computed in less than 80s. The LP-relaxation of DDT produces the best lower bounds, followed by DT. The DDT formulation slightly improves the best lower bound proposed in ([Sahli et al., 2015]). The EP formulation also exhibits good results, while being very fast. In terms of exact solving, the four MILP formulations are able to solve the instances involving 10 and 20 events in less than 120s. But they cannot solve the large instances within 300s.

As a perspective, we think that these LP-based lower bounds can be improved by adding valid inequalities to our MILP formulations. We also plan to generate a new benchmark dedicated to the ERCPSP and harder than the one proposed by [Neumann and Schwindt, 2002]. Another perspective is to build a branch-and-bound method to solve the ERCPSP with an exact method.

Instance Generation

Sommaire

5.1 Introduction 95
5.2 Basic Concepts 96
5.3 Basic Data and Precedence Graph Generation 99
5.4 Resource Demand and Availability Generation 103
5.5 Hardness of ERCPSP Instances 104
5.6 Functional Description of the generator 107
5.7 Conclusion 109

5.1 Introduction

Until 1992, the testset of [Patterson, 1984] was the main benchmark for RCPSP. This benchmark was not generated using a systematic approach controlled by several graph and resource-based parameters. It was shown that all the instances of this benchmark without exception belong to a class of easy problems [Kolisch et al., 1995]. Therefore, the analysis of algorithms should be based on a benchmark generated systematically by a problem generator. The performance of the tested algorithms can then be evaluated depending on different problem measures.

The ProGen of [Kolisch et al., 1995] is an instance generator for RCPSP. Several graph measures such as the number of nodes, the graph complexity, the number of predecessors and successors of a node as well as parameters for the resource constraints can be specified. [Schwindt, 1996] developed ProGen/max which generates instances of RCPSP/max (RCPSP with minimal and maximal time lags). ProGen/max is based on the methodology of ProGen for the construction of precedence graph structures.

Until now, no benchmark has been proposed for the ERCPSP. The only benchmark which is the most appropriate to our problem is the one proposed by [Neumann and Schwindt, 2002] for the Project Scheduling Problem with Inventory Constraints. In this chapter, we describe an instance generator for the ERCPSP also based on the methodology of ProGen.

The remainder of this chapter is structured as follows. Section 5.2 is concerned with basic definitions of graph theory and several graph measures which are known from literature. In Section 5.3 we present two different approaches to construct cyclic graphs. Section 5.4 deals with the generation of resource constraints. In Section 5.7 we conclude the chapter.

5.2 Basic Concepts

The generation steps of an instance of ERCPSP can be summarized as follows:

1. Generation of the basic data (see Subsection 5.3.1).
2. Construction of the precedence graph (see Subsection 5.3.2).
3. Determination of minimal and maximal time lags between events (see Subsection 5.3.3).
4. Generation of resource constraints (see Section 5.4).

5.2.1 Basic Definitions

In this section, we give some basic definitions and results which are used in Section 5.3 for the generation of precedence graphs. For an introduction to the theory of graphs we refer to [Bondy and Murty, 1976]. let us introduce the following notation. The symbols refer to graph $G = (X, U)$.

We assume that graph G is simple, which means it contains no directed loops or parallel arcs.

Definition 5.1 (Adjancency matrix **A** of a graph)**.** *The adjacency matrix* **A** *of graph* G *is defined to be the* $|X| \times |X|$ *matrix* $(\mathbf{A}_{ij})_{i,j \in X}$ *with*

$$\mathbf{A}_{ij} = \begin{cases} 1, if (i,j) \in U \\ 0, otherwise \end{cases}.$$

Definition 5.2 (Indegree and outdegree of node $i \in X$)**.** *The indegree* $\delta^-(i)$ *of node* $i \in X$ *is defined to be the number of (direct) predecessors of node* i*:* $\delta^-(i) = |\Gamma^+(i)|$*. Analogously, the outdegree* $\delta^+(i)$ *of node* $i \in X$ *is defined to be the number of (direct) successors of node* i*:* $\delta^+(i) = |\Gamma^-(i)|$*.*

X	set of nodes
$X(G)$	set of nodes of graph G
U	set of arcs
$U(G)$	set of arcs of graph G
(i,j)	arc from node $i \in X$ to node $j \in X$
v_{ij}	weight of arc (i,j)
$\delta^+(i)$	outdegree of node $i \in X$
$\delta^-(i)$	indegree of node $i \in X$
$\Gamma^+(i)$	set of direct successors of node $i \in X$
$\Gamma^-(i)$	set of direct predecessors of node $i \in X$
$\mathbf{A}$	adjacency matrix
$\mathbf{R}$	reachability matrix
$\mathcal{R}(i)$	set of nodes $j \in X$ which are reachable from node $i \in X$
$\bar{\mathcal{R}}(i)$	set of nodes $j \in X$ which node $i \in X$ can be reached
$\mathcal{C}$	set of cycle structures
$C(i)$	cycle structure to which node $i \in X$ belongs

Definition 5.3 (Reachability). *A node $j \in X$ is called reachable from node i if $j = i$ or if there is a (directed) path from i to j.*

Definition 5.4 (Reachability matrix of a graph). *The reachability matrix* $\mathbf{R}$ *of graph $G = (X,U)$ is defined to be the $n \times n$ matrix $(\mathbf{R}_{ij})_{i,j \in X}$ with*

$$\mathbf{R}_{ij} \begin{cases} 1, \text{if } j \text{ is reachable from } i \\ 0, \text{otherwise} \end{cases} .$$

Definition 5.5 (Connectivity). *Let $G = (X,U)$ be a graph with reachability matrix* $\mathbf{R}$*. Two nodes $i,j \in X$ are called connected in G if $i = j$ or if there is a sequence $(i_0, i_1, ..., i_k)$ of nodes $i_s \in V$ $(s = 0, ..., k)$ with $i_0 = i, i_k = j$, and*

$$\prod_{s=1}^{k} [1 - (1 - \mathbf{R}_{i_{s-1},i_s}) * (1 - \mathbf{R}_{i_s,i_{s-1}})] = 1$$

Definition 5.6 (Subgraph induced by node set). *A graph $G' = (X',U')$ is a subgraph of a graph $G = (X,U)$ if $X' \subseteq X$, $U' \subseteq U$ and $((i,j) \in U' \Rightarrow i,j \in X')$.*
A graph $G'' = (X'',U'')$ is a subgraph of a graph $G = (X,U)$ induced by node set X'' if $X' \subseteq X$ and $((i,j) \in U'' \Leftrightarrow i,j \in X'', (i,j) \in U)$.

Definition 5.7 (Weak component). *A weak component $G'(X',U')$ of a graph $G = (X,U)$ is a maximal subgraph of G (with respect to $|X'|$) induced by node set X' for which all nodes $i,j \in X$ are connected.*
A graph G which constitutes a weak component of itself is called weakly connected.

Definition 5.8 (Network). *An arc-weighted graph $N = (X, U, v)$ is called network if the underlying graph $G = (X, U)$ is weakly connected.*

Definition 5.9 (Strong component). *A strong component $G' = (X', U')$ of G is a maximal subgraph of G (with respect to $|X'|$) for which all nodes $i, j \in X'$ are mutually reachable.*
A graph G which constitutes a strong component of itself is called strongly connected.

Definition 5.10 (Cycle structure). *A cycle structure $C(i) = (X', U')$ of G is a strong component of G with $|X'| \geq 2$.*

Definition 5.11 (Redundant arc). *An arc $(i, j) \in U$ is said to be redundant in $G = (X, U)$ if there is a (directed) path from i to j in G which contains more than one arc.*

5.2.2 Graph Measures

The structure of precedence graphs has generally an impact on the time which an exact algorithm requires for solving a scheduling problem. In literature, a large number of graph measures can be found which describe the size, the logic, and the shape of graphs [Thesen, 1977, Patterson, 1976, Kurtulus and Davis, 1982]. The following parameters are the most commonly used:

- Number of nodes.
- Thesen's estimator for the restrictiveness (see below).
- Degree of redundancy.
- Number of predecessors and number of successors of a node.
- Number of cycle structures.
- Number of backward arcs.
- Number of nodes in a cycle structure.

All these parameters are used by our generator for the generation of precedence graphs.

Definition 5.12 (Restrictiveness of a graph). *Let $G = (X, U)$ be a weakly connected graph with exactly one source and one sink and node set $X = \{0, 1, ..., n, n+1\}$. Let n_Π denote the number of permutations $(i_1, i_2, ..., i_n)$ of $X' = \{1, ..., n\}$ such that if $k < l$ then $i_k \notin \mathcal{R}(i_l)$. The restrictiveness is equal to $1 - \frac{\log n_\Pi}{\log n!}$.*

Note that the determination of n_Π is a hard combinatorial problem. That is why Thesen has tested a set of over 40 different estimators for the restrictiveness. We denote by RT the best obtained estimator.

$$RT = 1 - \frac{(n+2)(n+3) - 2\sum_{i,j \in X} \mathbf{R}_{ij}}{(n)(n-1)} \tag{5.1}$$

Theorem 5.1 ([Schwindt, 1996]). *Let $G = (X, U)$ be a weakly connected acyclic graph with exactly one source 0 and exactly one sink $n+1$, node set $X = \{0, 1, ..., n, n+1\}$, and restrictiveness estimator RT. For RT the following properties apply:*

1. *$RT \in [0, 1]$.*
2. *$RT = 0$ exactly if G is parallel.*
3. *$RT = 1$ exactly if G is serial.*
4. *The insertion of a non-redundant arc in G increases RT.*
5. *The insertion of a redundant arc in G does not affect RT.*

5.3 Basic Data and Precedence Graph Generation

5.3.1 Basic Data

The user of our generator has to enter values for the following basic data which will be used to generate the precedence graph and the resource constraints:

n^{min}, n^{max}: minimal and maximal number of events.

Q^{min}, Q^{max}: minimal and maximal number of nonrenewable resources.

Let $rand\{a, ..., b\}$ $(a, b \in \mathbb{Z})$ be an integer pseudo random number out of the set $\{a, ..., b\}$. The basic data is calculated as follows:

- number of events $n = rand\{n^{min}, ..., n^{max}\}$
- number of nonrenewable resources $|Q| = rand\{Q^{min}, ..., Q^{max}\}$

5.3.2 Precedence Graph

Let us consider a weakly connected graph $G = (X, U)$. The precedence constraints of ERCPSP instances are given by the graph G and the corresponding minimal and maximal time lags (arc weights). A minimal time lag T_{ij}^{min} between the occurrence time of event i and the occurrence time of event j is represented by an arc (i, j) weighted by $v_{ij} = T_{ij}^{min}$. A maximal time lag T_{ij}^{max} between the occurrence time of event i and the occurrence time of event j is represented by a backward arc (j, i) weighted by $v_{ji} := -T_{ij}^{max}$. If there is a minimal time lag $T_{ij}^{min} > 0$ and a maximal time lag $T_{ji}^{max} > 0$, then the maximal time lag T_{ji}^{max} is ignored. A precedence constraint concerning a single event does not make sense in project scheduling. Therefore, the generation of the precedence graph is limited to the case of simple graphs.

Algorithm 5.1: Direct method

1. Generation of an acyclic graph without redundancy;
   (a) Selection of sources and sinks (nodes which will correspond to initial and terminal events);
   (b) Generation of direct predecessors;
   (c) Generation of direct successors;
   (d) Insertion of additional arcs such that the resulting graph is still without redundancy;
2. Insertion of redundant arcs;
3. Generation of cycle structures;
   (a) Creation of cycle structures (Definition 5.13)
   (b) Extension of cycle structures (Definition 5.14)
   (c) Densification of cycle structures (Definition 5.15)
4. Addition of a supersource and a supersink

[Schwindt, 1996] considered two methods to generate cyclic graphs. The first algorithm, called direct method, begins with the generation of an acyclic graph. Then, backward arcs are added to generate cycle structures. Finally, a supersource and a supersink are added to obtain a weakly connected graph. The second algorithm, called contraction method, first creates cycle structures which are then contracted. With the contracted cycle structures and the nodes not employed during the first step we generate an acyclic graph similarly to the direct method (see

Algorithm 5.2: Contraction method

1. Generation of cycle structures
   (a) Generation of several weak components
   (b) Transformation of the weak components to cycle structures
2. Contraction of the cycle structures to contracted cycle structures (Definition 5.16)
3. Generation of an acyclic graph based on the contracted cycle structures and additional nodes
4. Expansion of the contracted cycle structures and insertion in the graph (Definition 5.17)
5. Addition of a supersource and a supersink

Algorithm 5.1). Subsequently, the contracted cycle structures are expanded and integrated into the graph. Finally, we add a supersource and a supersink to obtain a weakly connected graph (see Algorithm 5.2). Efficient algorithms for the direct method and the contraction method are provided in [Schwindt, 1996].

Definition 5.13 (Creation of a cycle structure within a graph). *Let $G = (X, U)$ be a graph with set of cycle structures $\mathcal{C}$. By a creation of a cycle structure $C(i)$ within G we mean an operation on G which derives a graph $G' = (X', U')$ with set of cycle structures $\mathcal{C}'$ such that $\mathcal{C} \subset \mathcal{C}'$, $U \subset U'$, and $|U'| = |U| + 1$.*

Definition 5.14 (Extension of a cycle structure within a graph). *Let $G = (X, U)$ be a cyclic graph with set of cycle structures $\mathcal{C} \neq \emptyset$. By an extension of a cycle structure $C(i)$ within G we mean an operation on G which derives a graph $G' = (X', U')$ with set of cycle structures $\mathcal{C}'$ such that $|\mathcal{C}| \subset |\mathcal{C}'|$, $U \subset U'$, $|U'| = |U|+1$, and $\exists C'(i) \in \mathcal{C}'$ with $X(C(i)) \subset X(C'(i))$.*

Definition 5.15 (Densification of a cycle structure within a graph G). *Let $G = (X, U)$ be a cyclic graph with set of cycle structures $\mathcal{C} \neq \emptyset$. By a densification of a cycle structure $C(i)$ within G we mean an operation on G which derives a graph $G' = (X', U')$ with set of cycle structures $\mathcal{C}'$ such that $|\mathcal{C}| \subset |\mathcal{C}'|$, $U \subset U'$, $|U'| = |U| + 1$, and $X(C(i)) = X(C'(i)), \forall C'(i) \in \mathcal{C}'$.*

Definition 5.16 (Contraction of a cycle structure [Schwindt, 1996]). *The contraction of a cycle structure C to a contracted cycle structure c in a graph $G = (X, U)$*

is an operation on G which derives a graph $G' = (X', U')$ such that:

$$\begin{aligned} X' &= X \setminus X(C) \cup \{c\} \\ U' &= U \setminus \{(i,j) \in U \mid \{i,j\} \cap X(C) \neq \emptyset\} \\ &\quad \cup \{(c,j) \mid \exists (i,j) \in U, j \in X(C)\} \\ &\quad \cup \{(i,c) \mid \exists (i,j) \in U, i \in X(C)\} \end{aligned}$$

Definition 5.17 (Expansion of a contracted cycle structure)**.** *Let c be a contracted cycle structure of C in a graph $G = (X, U)$ and Let $G' = (X', U'))$ be a subgraph of G. By the expansion of c with respect to G' we mean an operation on G which derives a graph $G'' = (X'', U''))$ such that*

$$\begin{aligned} X' &= X \setminus (X(C) \cap X') \cup \{c\} \\ U' &= \{(i,j) \in U' \mid \{i,j\} \subseteq X''\} \setminus \{(i,j) \in U \mid c \in \{i,j\}\} \end{aligned}$$

The following input data are required for the generation of acyclic graphs:

- X: set of nodes.
- $NB_{n+1}^{min}, NB_{n+1}^{max}$: minimum and maximum number of sinks in G.
- NB_0^{min}, NB_0^{max}: minimum and maximum number of sources in G.
- NB_r^-: maximum number of non-redundant arcs entering node $i \in X$.
- NB_r^+: maximum number of non-redundant arcs entering node $i \in X$.
- RT: restrictiveness of Thesen for acyclic weak components.
- ρ: degree of redundancy in G.

To generate cycle structures in an acyclic graph $G = (X, U)$, we need the following input data:

- MLT^{min}, MLT^{max}: minimum and maximum percentage of maximum time lags.
- CS^{min}, CS^{max}: minimum and maximum number of cycle structures, respectively.
- n_c^{min}, n_c^{max}: minimum and maximum cardinal number of a cycle structure.
- δ: percentage of arcs employed for cycle structure densification.

5.3.3 Minimal and maximal time lags

Let $G = (X, U)$ be a weakly connected graph generated in the previous section. All the forward arcs of G belong to minimal time lags and all the backward arcs, which were added to define cyclic structures, belong to maximal time lags. To represent the precedence constraints by the arc-weighted graph $N = (X, U, v)$ we have to introduce a weight v_{ij} for any arc $(i, j) \in U$. The following input data have to be specified:

- D^{min}, D^{max}: minimal and maximal value of minimal time lags.
- $CST \in [0, 1]$: cycle structure tightness.
- $SF \in [0, +\infty[$: slack factor.

The minimal time lags T_{ij}^{min} are generated randomly out of the set $\{D^{min}, ..., D^{max}\}$: $T_{ij}^{min} = rand\{D^{min}, ..., D^{max}\}$. Let $N' = (X, U', v')$ be an acyclic arc-weighted graph such that U' is the subset of U that contains all arcs corresponding to minimal time lags. We denote by $\Gamma'^{+}(i)$ the set of direct successors of node i in N'. Let l'_{ij} be the length of the longest path from i to j in N', and let ω'_{ij} maximal time lag which can always be met:

$$\omega'_{ij} = \sum_{k \in \mathcal{R}(i) \cap \bar{\mathcal{R}}(i) \setminus \{j\}} \max_{l \in \Gamma'^{+}(k) \cap \bar{\mathcal{R}}(j)} \{v_{kl}\} \tag{5.2}$$

The maximal time lags T_{ij}^{max} are determined randomly in interval [Schwindt, 1996]:

$$[\omega'_{ij} - (\omega'_{ij} - l'_{ij})CST^2, (\omega'_{ij} - 2 * (\omega'_{ij} - l'_{ij})CST + (\omega'_{ij} - l'_{ij})CST^2)(1 + SF)].$$

5.4 Resource Demand and Availability Generation

In the following, we present the generation of resource requirements and resource availability.

5.4.1 Resource Demand

The following input data are required for the generation of resource constraints:

- a^{min}, a^{max}: minimal and maximal resource requirement.
- RF: Resource factor.

The resource constraints are generated as follows. First, for each pair $(i,k) \in X \times Q$, we determine whether or not event i produces or consumes resource k ($a_i^k \neq 0$). The resource factor RF determines the percentage of pairs (i,k) belonging to a requirement. The resource requirements a_i^k are then generated randomly from the set $\{a^{min}, ..., a^{max}\}$.

5.4.2 Resource Availability

The following input data is required for the generation of resource availability: Resource Strength (RS). RS controls the scarcity of resources. The initial availability of resource Q_k is given by

$$Q_k = RS * (\min_{t \geq 0}\{\sum_{i \in X, ES_i \leq t} a_i^k\}) + (1 - RS) * \sum_{i \in X} a_i^k.$$

$RS = 0$ implies that the resource levels of resource-feasible schedules are constant over time, whereas for $RS = 1$ the earliest schedule ES is feasible and thus optimal.

5.5 Hardness of ERCPSP Instances

In order to evaluate the impact of the generator control parameters on the hardness of the Extended RCPSP, we have generated 1620 instances of ERCPSP. We have used a full factorial design for four parameters which have proven to be closely related to the hardness ERCPSP instances. Table 5.1 shows the constant parameter values and Table 5.2 provides the variable parameter values.

Q^{min}	Q^{max}	NB_{n+1}^{min}	NB_{n+1}^{max}	NB_0^{min}	NB_0^{max}	NB_r^-	NB_r^+	ρ	δ
5	5	1	5	1	5	3	3	0.05	0.5
MLT^{min}	MLT^{max}	CS^{min}	CS^{max}	n_c^{min}	n_c^{max}	D^{min}	D^{max}	CST	SF
0.05	0.25	1	5	2	5	0	30	0.5	0

Tableau 5.1 – Constant parameter values

n	RT	RS	RF
10	0.25	0.00	0.50
20	0.50	0.25	0.75
30	0.75	0.50	1.00

Tableau 5.2 – Variable parameter values

For each combination of n, RT, RS and RF we have generated 20 instances of ERCPSP. 963 of the 1620 generated instances are feasible. We have used our branch-and-bound method to solve these instances (see Section 4.3). We have determined for each instance the computation time for the generation of an optimal solution (CPU_{fd}) and the computation time for the verification of optimality (CPU_{ver}), for which we imposed a time limit of 30 seconds.

All the feasible instances were solved to optimality within the time limit. Each best solution has been determined within $0.01s$. The verification of optimality, however, required a much larger mean computation time of $0.09s$. Table 5.3 shows the effect of increasing the number of events on the computation times. As we can see, the computation times increase by increasing the size of the problem.

n	CPU_{ver}	CPU_{fd}
10	0.00	0.00
20	0.02	0.00
30	0.09	0.01

Tableau 5.3 – Effects of number of events n on problem hardness

The impact of varying the restrictiveness of the precedence graph can be seen in Figure 5.1. As we can see, the restrictiveness of the precedence graph has a strong impact on the hardness of ERCPSP instances. In fact, computation times increase with decreasing the restrictiveness of precedence graphs.

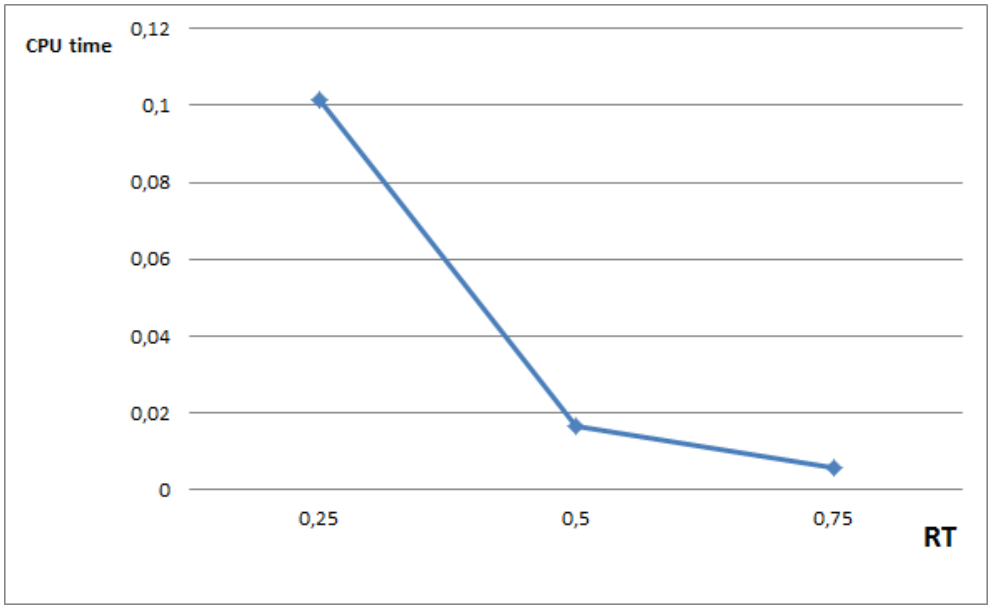


Figure 5.1 – Effects of the restrictiveness RT on problem hardness

Figure 5.2 provides the impact of increasing the resource factor (RF). The computation times increase with increasing the resource factor. In fact, a large resource factor increases the number of resource conflicts. Each resource conflict generates several nodes in the search tree of our branch-and-bound method.

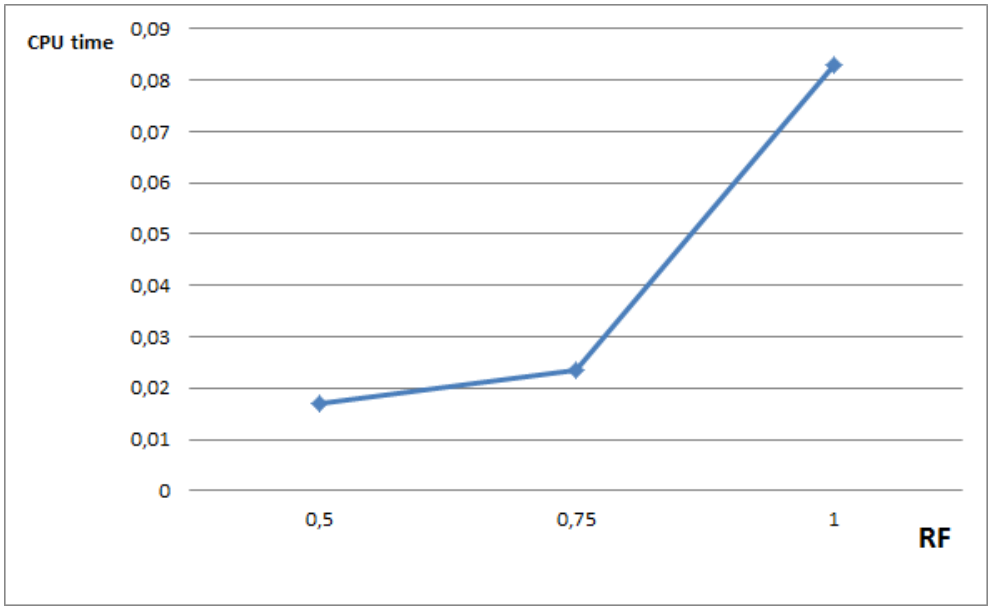


Figure 5.2 – Effects of the resource factor RF on problem hardness

Figure 5.3 provides the impact of decreasing the resource strength (RS). An interesting relationship between computation times and scarcity of resources (RS) is provided. A decrease in resource availability makes the problems much more harder. In fact, a small resource availability increases the number of resource conflicts which increases the nodes in the search tree.

Note that 333 among 360 instances of [Neumann and Schwindt, 2002] were generated with a restrictiveness larger than 0.5 and a resource strength larger than 0.2. This is why the instances are easy to solve.

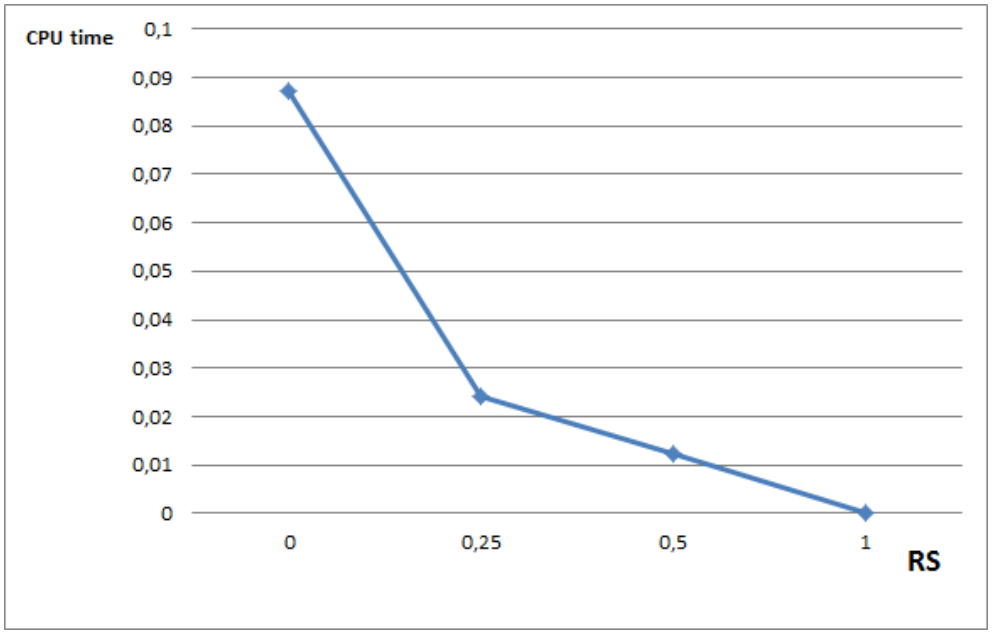


Figure 5.3 – Effects of the resource strength RS on problem hardness

5.6 Functional Description of the generator

Our generator was coded in java and C++ language. Figure 5.4 shows the menu control of this instance generator.

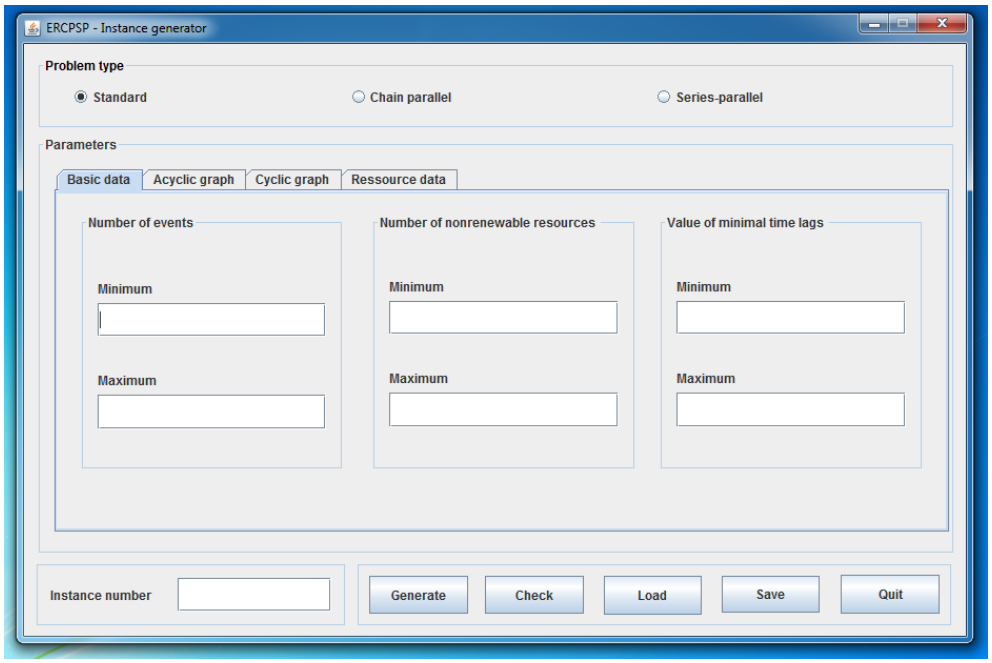


Figure 5.4 – Genrator menu control

First, the type of the ERCPSP instance (standard problem, chain parallel case

or series-parallel case) must be fixed. Then, the control parameters defined in the previous section have to be set using manual values, the different parameters are shown in Figures 5.5-5.8. Finally, the number of instances with the same parameters has to be specified.

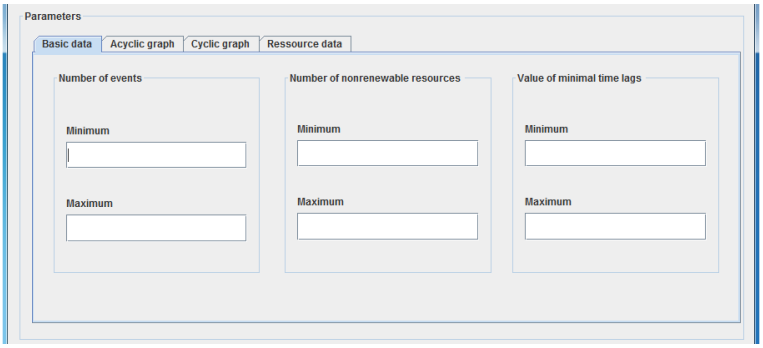


Figure 5.5 – Basic data parameters

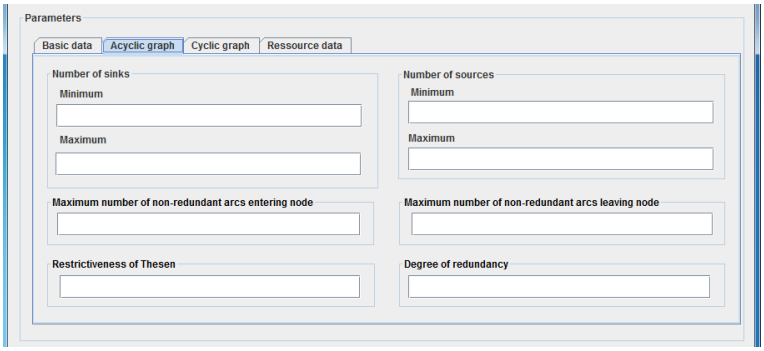


Figure 5.6 – Acyclic graph parameters

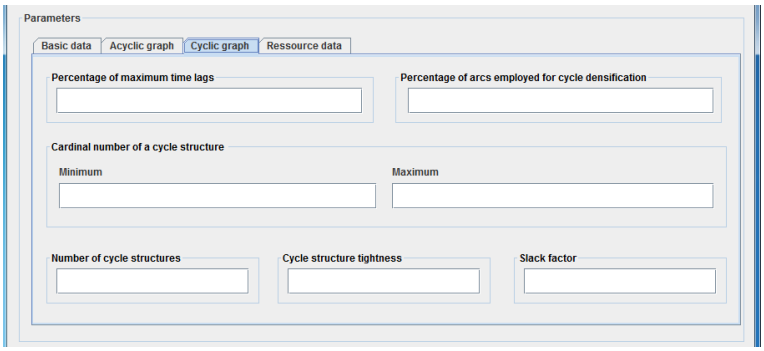


Figure 5.7 – Cyclic graph parameters

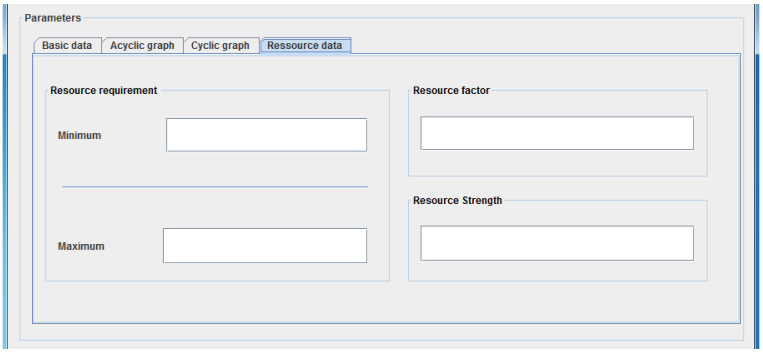


Figure 5.8 – Ressource parameters

The following actions could be performed:

- **Generate:** Begins the generation of ERCPSP instances corresponding to the specified parameters.
- **Check:** Verifies the consistency conditions for the parameter values.
- **Load:** Sets the values in the control menu to a previously saved parameters.
- **Save:** Saves the current parameters in a text file.
- **Quit:** Exit the generator.

This generator creates two type of files during the generation.

- ***.shl:** contains an instance of ERCPSP.
- **Stat.txt:** statistics file that involves several instance characteristics (control parameters, lower bounds,...).

5.7 Conclusion

In this chapter, we have developed an instance generator for the ERCPSP. This generator takes into account several graph measures such as the number of nodes, the graph complexity, the number of predecessors and successors of a node as well as parameters for the generation of the basic data and the resource constraints. It is based on the methodology of ProGen [Kolisch et al., 1995].

Conclusion and perspectives

In this thesis, we are interested in proposing new methodologies and approaches to solve ERCPSP. This problem is a general scheduling problem where the availability of resources is depleted and replenished [Carlier et al., 2009, Carlier et al., 2016]. An instance of ERCPSP consists of events, nonrenewable resources and generalized precedence constraints between pairs of events. Each event produces or consumes some units of resources at its occurrence time. The objective is to build a schedule that satisfies the precedence and resource constraints and minimizes the makespan. ERCPSP is a generalization of RCPSP where activities requiring renewable resources are replaced by events consuming or producing nonrenewable resources.

After introducing the field of the study of the thesis in Chapter 1, we have presented in Chapter 2 some terminology dedicated to basic concepts and formulate the ERCPSP, and we have shown the connection between this problem and other scheduling problems with production and consumption of resources. After that, we have studied four special cases of ERCPSP for which the decision problem can be solved in polynomial time: the relocation problem, the parallel chain case, the series-parallel case and the interval order case. Finally, we have presented a dynamic programming algorithm to solve the parallel chain case and approximate methods based on the concept of linear orders.

Chapter 3 investigates lower bound techniques. We have proposed six lower bounds for ERCPSP. Two of them are based on the extraction of a generalized Cumulative Scheduling Problem, combined with an adapted version of Jackson's Pseudo-Preemptive Schedule [Carlier and Pinson, 2004] and the concept of energetic reasoning. Two further lower bounds respectively result from applying Carlier and Rinnooy Kan's Shifting Algorithm to a Financing Problem and iteratively testing the feasibility of associated network flow problems in a dichotomic search method. The last two lower bounds are destructive lower bounds computed using a general linear programming scheme.

Chapter 4 deals with the exact solving of ERCPSP. In the first half of the chapter, we have introduced four mixed integer linear programming formulations for ERCPSP. In the second half of the chapter, we have presented a branch-and-bound

method to solve the ERCPSP. We have also adapted two constraint propagation algorithms to improve our method: the timetabling of [Le Pape, 1994] and the balance constraint of [Laborie, 2002].

Finally in Chapter 5, we have developed an instance generator for ERCPSP based on the methodology of ProGen [Kolisch et al., 1995]. This generator takes into account several graph measures such as the number of nodes, the graph complexity, the number of predecessors and successors of a node as well as parameters for the generation of the basic data and the resource constraints.

Perspectives

The contributions in this thesis clear the way for numerous perspectives to be done. We provide in the following the ongoing work as well as the future openings we plan to investigate.

Approximate methods for ERCPSP. We can associate an earliest schedule with a consumption linear order. So an exact method would be the list-based scheduling algorithm which enumerates all consumption linear orders. But if it is too costly, we have to deal with approximate methods. We now propose a straightforward generalization of the list algorithm introduced for scheduling problems with renewable resources. We first choose a priority function on the event set. The list algorithm schedules events at certain decision times. These decision times are $t = 0$ and the available times of events. For every time t, the algorithm chooses the event with the highest priority from among all unscheduled ready events and schedules it at time t. This is repeated until no further events can be started at time t, then t is adjusted to the time where an event becomes available, unless all events are scheduled. We can also use a list of consumption events. In this case we only have consumption events in the priority list and the algorithm is the same as the previous list algorithm, except that we schedule all production events when they are ready. The list algorithm can be applied to graphs with non-negative arc weights and without directed cycles.

A Benchmark For ERCPSP. Until now, no benchmark has been proposed for the ERCPSP. The only benchmark which is the most appropriate to ERCPSP is the one proposed by Neumann and Schwindt. This encourages us to investigate providing the community with a set of instances for the problem. Using the instance generator described in Chapter 5, we aim to generate a new benchmark specially dedicated for ERCPSP. The set of instances will be made online for the researchers.

Column Generation. The difficulty encountered using the mixed integer programming models we proposed in Chapter 4 incites us to investigate a column

generation approach for ERCPSP.

References

[Abdel-wahab and Kameda, 1978] Abdel-wahab, H. and Kameda, T. (1978). Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. *Operations Research*, 26(1):141–158.

[Agin, 1966] Agin, N. (1966). Optimum seeking with branch and bound. *Management Science*, 13(4):176–185.

[Ahuja et al., 1995] Ahuja, R., Magnanti, T., and Orlin, J. (1995). Network flows. In Nemhauser, G., Kan, A. R., and Todd, M., editors, *Handbooks in Operations Research and Management Science*, pages 258–263. Elsevier, Amsterdam.

[Alvarez-Valdés and Goerlich, 1993] Alvarez-Valdés, R. and Goerlich, J. T. (1993). The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204 – 220.

[Applegate and Cook, 1991] Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156.

[Artigues et al., 2003] Artigues, C., Michelon, P., and Reusser, S. (2003). Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249 – 267.

[Balas, 1968] Balas, E. (1968). Project scheduling with resource constraints. Technical report, DTIC Document.

[Baptiste et al., 1999] Baptiste, P., Le Pape, C., and Nuijten, W. (1999). Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92(0):305–333.

[Bartusch et al., 1988] Bartusch, M., Möhring, R., and Radermacher, F. (1988). Scheduling project network with resource constraints and time windows. *Annals of Operations Research*, 16:201–240.

[Beck, 2002] Beck, J. (2002). Heuristics for constraint-directed scheduling with inventory. *Journal of Scheduling*, 5:45–69.

[Berger et al., 1992] Berger, I., Bourjolly, J.-M., and Laporte, G. (1992). Branch-and-bound algorithms for the multi-product assembly line balancing problem. *European Journal of Operational Research*, 58(2):215 – 222. Practical Combinatorial Optimization.

[Bondy and Murty, 1976] Bondy, J. A. and Murty, U. S. R. (1976). *Graph theory with applications*, volume 290. Citeseer.

[Bouly et al., 2005] Bouly, H., Carlier, J., Moukrim, A., and Russo, M. (2005). Solving rcpsp with resources production possibility by tasks. In *MHOSI'2005, 24-26 Avril 2005, Hammamet, Tunisie*.

[Brčić et al., 2012] Brčić, M., Kalpić, D., and Fertalj, K. (2012). Resource constrained project scheduling under uncertainty: a survey. In *23rd Central European Conference on Information and Intelligent Systems*.

[Brucker, 2007] Brucker, P. (2007). *Scheduling algorithms*, volume 3. Springer.

[Brucker et al., 1999] 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*, 112:3–41.

[Brucker and Knust, 2000] Brucker, P. and Knust, S. (2000). A linear programming and constraint propagation-based lower bound for the {RCPSP}. *European Journal of Operational Research*, 127(2):355 – 362.

[Carlier, 1984] Carlier, J. (1984). *Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité, Doctorat d'état, PhD thesis*. Université Pierre et Marie Curie (Paris VI).

[Carlier et al., 2016] Carlier, J., Moukrim, A., and Sahli, A. (2016). Lower bounds for the event scheduling problem with consumption and production of resources. *Discrete Applied Mathematics*, pages –.

[Carlier et al., 2009] Carlier, J., Moukrim, A., and Xu, H. (2009). The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm. *Discrete Appl. Math.*, 157(17):3631–3642.

[Carlier and Néron, 2003] Carlier, J. and Néron, E. (2003). On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149(2):314 – 324. Sequencing and Scheduling.

[Carlier and Pinson, 1998] Carlier, J. and Pinson, E. (1998). Jackson's pseudo preemptive schedule for the Pm/r_i, p_i/C_{max} scheduling problem. *Annals of Operations Research*, 83(0):41–58.

[Carlier and Pinson, 2004] Carlier, J. and Pinson, E. (2004). Jackson's pseudo-preemptive schedule and cumulative scheduling problems. *Discrete Applied Mathematics*, 145(1):80 – 94.

[Carlier and Rinnooy Kan, 1982] Carlier, J. and Rinnooy Kan, A. H. G. (1982). Scheduling subject to nonrenewable-resource constraints. *Oper. Res. Lett.*, 1(2):52–55.

[Cesta et al., 2002] Cesta, A., Oddi, A., and Smith, S. (2002). A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136.

[Chen et al., 2010] Chen, W., jun Shi, Y., fei Teng, H., ping Lan, X., and chen Hu, L. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, 180(6):1031 – 1039. Special Issue on Modelling Uncertainty.

[Christofides et al., 1987] Christofides, N., Alvarez-Valdes, R., and Tamarit, J. (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262 – 273.

[Debels and Vanhoucke, 2007] Debels, D. and Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469.

[Demeulemeester and Herroelen, 1990] Demeulemeester, E. and Herroelen, W. (1990). A branch-and-bound procedure for the multiple constrained-resource project scheduling problem. In *Proceedings of the Second International Workshop*

on Project Management and Scheduling, pages 8–25, Université de Technologie de Compiègne, Compiègne (France).

[Erik L. Demeulemeester, 1997] Erik L. Demeulemeester, W. S. H. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492.

[Erschler et al., 1991] Erschler, J., Lopez, P., and Thuriot, C. (1991). Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement. *Revue d'intelligence artificielle*, 5(3):7–32.

[Gass, 2003] Gass, S. (2003). *Linear Programming: Methods and Applications*. Dover Books on Computer Science Series. Dover Publications.

[Jackson, 1956] Jackson, J. R. (1956). An extension of johnson's results on job idt scheduling. *Naval Research Logistics Quarterly*, 3(3):201–203.

[Johannes, 2005] Johannes, B. (2005). On the complexity of scheduling unit-time jobs with or-precedence constraints. *Operations Research Letters*, 33:587–596.

[Johnson, 1954] Johnson, S. (1954). Optimal two and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68.

[Kaplan and Amir, 1988] Kaplan, E. and Amir, A. (1988). A fast feasibility test for relocation problems. *European Journal of Operational Research*, 35:201–205.

[Kolisch and Hartmann, 2006] Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37.

[Kolisch and Padman, 2001] Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, 49(3):249–272.

[Kolisch and Sprecher, 1997] Kolisch, R. and Sprecher, A. (1997). PSPLIB - a project scheduling problem library: OR software - ORSEP operations research software exchange program. *European Journal of Operational Research*, 96(1):205 – 216.

[Kolisch et al., 1995] Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41(10):1693–1703.

[Koné et al., 2011] Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2011). Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3 – 13. Project Management and Scheduling.

[Koné et al., 2013] Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2013). Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Flexible Services and Manufacturing Journal*, 25(1-2):25–47.

[Kurtulus and Davis, 1982] Kurtulus, I. and Davis, E. (1982). Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172.

[Laborie, 2002] Laborie, P. (2002). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artif. Intell.*, 143(2):151–188.

[Lahrichi, 1982] Lahrichi, A. (1982). Ordonnancements. La notion de "parties obligatoires" et son application aux problèmes cumulatifs. *RAIRO-Operations Research-Recherche Opérationnelle*, 16(3):241–262.

[Lawler et al., 1993] Lawler, E. L., Lenstra, J. K., Kan, A. H. R., and Shmoys, D. B. (1993). Chapter 9 sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445 – 522. Elsevier.

[Le Pape, 1994] 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.

[Lee et al., 1997] Lee, C.-Y., Lei, L., and Pinedo, M. (1997). Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41.

[Lgelmund and Radermacher, 1983] Lgelmund, G. and Radermacher, F. J. (1983). Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13(1):29–48.

[Lin and Cheng, 1999] Lin, B. and Cheng, T. (1999). Relocation problems to minimize the maximum tardiness and the number of tardy jobs. *European Journal of Operational Research*, 116:183–193.

[Lopez et al., 1992] Lopez, P., Erschler, J., and Esquirol, P. (1992). Ordonnancement de tâches sous contraintes: une approche énergétique. *Automatique-productique informatique industrielle*, 26(5-6):453–481.

[Mingozzi et al., 1998] Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Manage. Sci.*, 44(5):714–729.

[Moukrim et al., 2015] Moukrim, A., Quilliot, A., and Toussaint, H. (2015). An effective branch-and-price algorithm for the preemptive resource constrained project scheduling problem based on minimal interval order enumeration. *European Journal of Operational Research*, 244(2):360–368.

[Neumann and Schwindt, 2002] Neumann, K. and Schwindt, C. (2002). Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56(3):513–533.

[Neumann et al., 2005] Neumann, K., Schwindt, C., and Trautmann, N. (2005). Scheduling of continuous and discontinuous material flows with intermediate storage restrictions. *European Journal of Operational Research*, 165:495–509.

[Neumann et al., 2003] Neumann, K., Schwindt, C., and Zimmermann, J. (2003). *Project Scheduling with Time Windows and Scarce Resources*. Springer, Berlin.

[Neumann et al., 2006] Neumann, K., Schwindt, C., and Zimmermann, J. (2006). Resource-constrained project scheduling with time windows: Recent developments and new applications. In Jozefowska, J. and Weglarz, J., editors, *Perspectives in Modern Project Scheduling*, pages 375–407. Kluwer, Boston.

[Neumann and Zhan, 1997] Neumann, K. and Zhan, J. (1997). Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints. *Journal of Intelligent Manufacturing*, 19:205–217.

[Palem and Simons, 1993] Palem, K. V. and Simons, B. B. (1993). Scheduling time-critical instructions on risc machines. *ACM Trans. Program. Lang. Syst.*, 15(4):632–658.

[Papadimitriou and Yannakakis, 1979] Papadimitriou, C. H. and Yannakakis, M. (1979). Scheduling interval-ordered tasks. *SIAM Journal on Computing*, 8(3):405–409.

[Patterson, 1976] Patterson, J. H. (1976). Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics Quarterly*, 23(1):95–123.

[Patterson, 1984] Patterson, J. H. (1984). A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Manage. Sci.*, 30(7):854–867.

[Pinedo, 2012] Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

[Pritsker et al., 1969] 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(1):pp. 93–108.

[Ramamurthy, 2007] Ramamurthy, P. (2007). *Operations Research*. New Age International (P) Limited.

[Sahli et al., 2015] Sahli, A., Carlier, J., and Moukrim, A. (2015). Lower bounds for scheduling problems with production and consumption of resources. In *Seventh Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2015, August 25-28, 2015, Prague, Czech Republic*, pages 680–682.

[Scholl et al., 1997] Scholl, A., Klein, R., and Jurgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627 – 645.

[Schwindt, 1996] Schwindt, C. (1996). Generation of resource-constrained project scheduling problems with minimal and maximal time lags. Technical report, Institut fur Wirtschaftstheorie und Operations Research, Universitat.

[Selle, 1999] Selle, T. (1999). Lower bounds for project scheduling problems with renewable and cumulative resources. Technical report, WIOR, Universität Karlsruhe, Karlsruhe, Germany.

[Sethi, 1975] Sethi, R. (1975). Complete register allocation problems. *SIAM Journal of Computer*, 4(3):226–248.

[Slowiński, 1984] Slowiński, R. (1984). Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. *European Journal of Operational Research*, 15(3):366–373.

[Smith, 1956] Smith, W. (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3(1):59–66.

[Sourd, 2005] Sourd, F. (2005). Continuous filling and emptying of storage systems in constraint-based scheduling. *European Journal of Operational Research*, 165:510–524.

[Sprecher, 2000] Sprecher, A. (2000). Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science*, 46(5):710–723.

[Stinson et al., 1978] Stinson, J., Davis, E., and Khumawala, B. (1978). Multiple resource-constrained scheduling using branch-and-bound. *AIIE Transactions*, 10:252–259.

[Stork and Uetz, 2005] Stork, F. and Uetz, M. (2005). On the generation of circuits and minimal forbidden sets. *Mathematical programming*, 102(1):185–203.

[Thesen, 1977] Thesen, A. (1977). Measures of the restrictiveness of project networks. *Networks*, 7(3):193–208.

[Valdes et al., 1982] Valdes, J., Tarjan, R., and Lawler, E. (1982). The recognition of series-parallel digraphs. *SIAM Journal of Computer*, 11:298–313.

[Valls et al., 2005] Valls, V., Ballestín, F., and Quintanilla, S. (2005). Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, 165(2):375 – 386. Project Management and Scheduling.