Resource-Constrained Project Scheduling
Models, algorithms, extensions and applications

Christian ARTIGUES
Sophie DEMASSEY
Emmanuel NÉRON
(editors)

November 20, 2007

Contents

Preface

Nowadays, material and human resource management becomes an increasingly major issue for organizations. Thus, a careful management of projects is an absolute necessity to preserve the competitiveness of companies. Scheduling plays a major part in project management. Indeed, the scheduling process amounts to deciding when the project activities will start and how they will use the available resources. Such decisions can be expected to have a large impact on the project total duration (makespan), which is the main performance criterion we consider in this book. Note that establishing a good schedule is necessary but not sufficient for actual project makespan minimization. As a comprehensive counterexample, let us consider the project that consisted of writing the present book. We expected a one year total duration while we must admit that the realized makespan largely exceeded two years. In this case, the makespan increase was clearly caused by an over-optimistic estimation of activity durations and resource availabilities while the schedule was correct.

Nevertheless, we focus in this book on the deterministic resource-constrained project scheduling problem (RCPSP). The standard RCPSP can be defined as a combinatorial optimization problem, i.e. in terms of decision variables, constraints and objective function, as follows. A set of activities and a set of resources of known characteristics (activity durations, activity resource demands, resource availabilities, precedence restrictions) are given. The decision variables are the activity start times defined on integer time periods. The objective function which has to be minimized is the makespan, i.e. the largest activity completion time, assuming the project starts at time 0. There are two types of constraints. The precedence constraints prevent each activity from starting before the completion of its predecessors. The resource constraints ensure that, at each time period and for each resource, the total activity demand does not exceed the resource availability. Once started, an activity cannot be interrupted.

Despite the simplicity of its definition, the RCPSP belongs to the class of NP-hard optimization problems [BLA 83] and is actually one of the most intractable classical

problems in practice. In the publicly available PSPLIB library [PSPib], the optimal makespan of randomly generated problems with 60 activities and 4 resources is still unkown. For both its industrial relevance and its challenging difficulty, solving the RCPSP has become a flourishing research theme. This becomes clear when observing the significant number of books that were published on this subject [SLO 89, KOL 95a, WEG 98, HAR 99, KLE 01, DOR 02, DEM 02c, NEU 03, BRU 06, JOZ 98]. One might even legitimately ask why yet another book on the RCPSP is needed.

The aim of this book is to present the latest results in the field, not as a collection of independent articles but as a whole, in an integrated manner. Thus, it provides a state-of-the-art of the solution approaches for the standard RCPSP (Part I) and for several of its variants (Part II) and industrial applications (Part III). The developments on each approach type are investigated from the fundamentals to the most recent – even current – studies.

This book contains 18 chapters written by 28 authors which are all active researchers in the area. We describe herafter the contents of the book, underlining the original aspects for each chapter.

There are three parts. The first part is devoted to the standard RCPSP as stated above and to the various methods that have been proposed to solve it. More precisely, each chapter of this part focuses on a particular aspect of the solution process. Chapter 1, written by Christian Artigues, focuses on the models and describes the RCPSP as a combinatorial optimization problem. Different combinatorial models of the resource constraints and characterizations of dominant schedule sets are given. In particular, the properties of the order-based representation are studied. Chapter 2, written by Emmanuel Néron, considers the scheduling standard subproblems obtained by various relaxations. Solving these subproblems through specific scheduling methods generally yields quickly computable lower bounds. However, extraction of a relevant subproblem may be a complex task. The chapter describes innovative concepts of redundant resources and redundant functions, designed to this aim. Chapter 3, written by Sophie Demassey, is devoted to mathematical programming formulations of the RCPSP and to lower bounds resulting from their relaxations. Besides recalling the main sequence-based and time-indexed integer linear programming formulations, it describes the recent cutting-plane and column generation techniques allowing to obtain tight lower bounds when used in conjunction with constraint programming. A promising formulation based on the redundant resource concept and linear lower bounds is also introduced. Chapter 4, written by Philippe Laborie and Wim Nuijten, is devoted to constraint programming approaches for scheduling and describes the existing propagation algorithms relevant for the RCPSP. The by-now standard disjunctive, edge-finding and energy reasoning rules are recalled. The original balance and energy-precedence propagation algorithms are presented. Chapter 5, written by Emmanuel Néron, presents the main branching schemes and associated dominance

rules used to solve the RCPSP via exact methods. Indeed, a multitude of branch-and-bound methods can be derived by combining the presented branching schemes with the lower bounds and constraint propagation algorithms presented in Chapters 2, 3 and 4. Chapter 6, written by Christian Artigues and David Rivreau, describes the main heuristics and metaheuristics that were proposed to approximate the RCPSP. The chapter reviews the literature and proposes a unifying framework based on the concepts of events and resource flows for the two main constructive algorithms : the parallel and the serial scheduling schemes. Chapter 7, written by Christian Artigues, Oumar Koné, Pierre Lopez, Marcel Mongeau, Emmanuel Néron and David Rivreau, is devoted to extensive computational experiments. Its objective is twofold. First, a selection of representative exact and heuristic methods among the ones presented in the previous chapters are tested and compared under a common experimental framework on four different instance sets. Second, classical and new instance difficulty indicators are evaluated through the experiments and their discriminating power is discussed.

The second part of the book presents variants and extensions of the RCPSP. Clearly, the standard model does not allow for modeling accurately all the practical situations. Each chapter of this part focuses on a particular variant or extension and presents the latest advances to tackle it. Most chapters present new and original methods. Chapter 8, written by Jean Damay, presents LP-based neighborhood search and exact methods for the rational preemptive variant of the RCPSP. This variant has never been considered before despite its practical interest. It considers the case where activities can be interrupted at any, non-necessarily integer, time. Chapter 9, written by Odile Bellenguez-Morineau and Emmanuel Néron, is devoted to the multi-mode RCPSP and the multi-skill RCPSP. The chapter reviews branching schemes and dominance rules for the multi-mode RCPSP. This extension introduces alternative activity execution modes, differing by the resource requirements and the durations. The multi-skill RCPSP is a variant for which the modes are implicitely defined through the intermediate concept of required and available skills. This model allows to efficiently represent the practical case of human resources while there would be an explosion of the number of modes in the multi-mode model. An exact method is proposed to solve the problem. Chapter 10, written by Jacques Carlier, Aziz Moukrim and Huang Xu, considers a RCPSP with renewable (as in the standard case) and non-renewable resources. Some activities consume a non-renewable resources while others produce it. They propose an enumeration method based on the concept of arbitrage and on the strict order algorithm. They also show how AND/OR precedence constraints can be used to compute solutions. Chapter 11, written by Christian Artigues and Cyril Briand, considers the RCPSP with minimum and maximum time lags. They focus on the problem of activity insertion in a schedule represented by a resource flow, which can occur in reactive scheduling (next chapter) or as a component of a local search method. They show the problem is NP-hard and propose a polynomial algorithm when only minimum time lags are considered. Chapter 12,

written by Christelle Guéret and Narendra Jussien, proposes a reactive approach to solve a dynamic project scheduling problem, i.e. that may change over time through unexpected events. Their method can tackle various events including the insertion of a new activity as in the previous chapter. They make use of explanation-based constraint programming, where explanations allow for schedule repairing through tree-search after each disruption. Chapter 13, written by Erik Demeulemeester, Willy Herroelen and Roel Leus, makes a step further in tackling project scheduling under uncertainty through proactive/reactive procedures. Uncertainty may affect activity durations and resource availabilities. Proactive scheduling methods aiming at anticipating disruptions through robust resource allocation and buffer insertion in a precomputed schedule are presented. Reactive strategies are carried out when the proactive schedule cannot absorb the disruptions. The objective is to minimize an unstability function measuring the distance between the precomputed schedule and the realized schedule. While the previous chapters mainly consider makespan minimization, Chapter 14, written by Laure-Emmanuelle Drezet, introduces several other objective functions dealing with financial aspects. The chapter reviews and classifies the existing work carried out on net present value maximization and resource-related costs such as resource levelling and resource investment.

The third part of the book is devoted to industrial applications involving the RCPSP or its variants and extensions. Thereby, the reader may verify how the different models and methods presented in this book are actually applied in various and maybe surprising situations. Chapter 15, written by Michel Gourgand, Nathalie Grangeon et Sylvie Norre, considers an assembly shop problem in an automotive company. Special attention is paid to the modeling process which exhibits a RCPSP with variable resource profile and resource individualization, linked with the multimode aspects described in chapter 9. The problem is solved by neighborhood search heuristics. Chapter 16, written by Laure-Emmanuelle Drezet and Jean-Charles Billaut, describes an employee scheduling problem in a IT company. The problem is modeled as a mixed project scheduling and workforce assignment problem. It follows that work regulation constraints are added to the standard RCPSP model. Since the considered projects (software developments) are highly likely to be disrupted, a proactive/reactive method (see chapter 13) is proposed. Chapter 17, written by Christoph Schwindt and Norbert Trautmann, considers a rolling ingots production scheduling problem in aluminium industry. This industrial context gives rise to a complex multi-mode RCPSP with minimum and maximum time lags, batching machines and sequence-dependent changeover times. A filtered beam search version of a branch-and-bound method is proposed to solve the problem. Chapter 18, written by Benoit Dupont de Dinechin, Christian Artigues and Sadia Azem, presents an unusual application of the RCPSP, namely an instruction scheduling problem for very long instruction word processors. The specific characteristics of these processors and the cyclic nature of the problem yields a modulo RCPSP with minimum and maximum time lags. New integer linear

programming formulations are proposed and an efficient large neighborhood search technique is used to solve near-optimally large industrial problems in a few minutes.

We wish to warmly thank all the colleagues who kindly accepted to review one or several chapters of the book: Philippe Baptiste, Rainer Kolisch, Philippe Lacomme, Francis Sourd and Mario Vanhoucke.

Christian Artigues, Sophie Demassey, Emmanuel Néron

FIRST PART

Models and Algorithms for the Standard Resource-Constrained Project Scheduling Problem

Chapter 1

The Resource-Constrained Project Scheduling Problem

1.1. A combinatorial optimization problem

Informally, a resource-constrained project scheduling problem (RCPSP) considers resources of limited availability and activities of known durations and resource requests, linked by precedence relations. The problem consists of finding a schedule of minimal duration by assigning a start time to each activity such that the precedence relations and the resource availabilities are respected.

More formally, the RCPSP can be defined as a combinatorial optimization problem. A combinatorial optimization problem is defined by a solution space $\mathcal{X}$, which is discrete or which can be reduced to a discrete set, and by a subset of feasible solutions $\mathcal{Y} \subseteq \mathcal{X}$ associated with an objective function $f : \mathcal{Y} \rightarrow \mathbb{R}$. A combinatorial optimization problem aims at finding a feasible solution $y \in \mathcal{Y}$ such that $f(y)$ is minimized or maximized. A resource-constrained project scheduling problem is a combinatorial optimization problem defined by a tuple $(V, p, E, \mathcal{R}, B, b)$.

Activities constituting the project are identified by a set $V = \{A_0, \ldots, A_{n+1}\}$. Activity A_0 represents by convention the start of the schedule and activity A_{n+1} symmetrically represents the end of the schedule. The set of non-dummy activities is identified by $\mathcal{A} = \{A_1, \ldots, A_n\}$.

Durations are represented by a vector p in $\mathbb{N}^{n+2}$ where p_i is the duration of activity A_i, with special values $p_0 = p_{n+1} = 0$.

Chapter written by Christian ARTIGUES.

Precedence relations are given by E, a set of pairs such that $(A_i, A_j) \in E$ means that activity A_i precedes activity A_j. A precedence activity-on-node graph $G(V, E)$ is defined where nodes correspond to activities and arcs correspond to precedence relations[1]. We assume that G contains no cycle, otherwise the precedence relations are obviously inconsistent. Since precedence is a transitive binary relation, the existence of a path in G from node A_i to node A_j also means that activity A_i precedes activity A_j. Thus, all precedence graphs having the same transitive closure define the same precedence constraints. We assume that, taking account of the preceding remark, E is such that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.

Renewable resources are formalized by set $\mathcal{R} = \{R_1, \ldots, R_q\}$.

Availabilities of resources are represented by a vector B in $\mathbb{N}^q$ such that B_k denotes the availability of R_k. In particular, a resource R_k such that $R_k = 1$ is called a unary or disjunctive resource. Otherwise, as a resource may process several activities at a time, it is called a cumulative resource.

Demands of activities for resources are abstracted by b, a $(n+2) \times q$ integer matrix, such that b_{ik} represents the amount of resource R_k used per time period during the execution of A_i.

A *schedule* is a point S in $\mathbb{R}^{n+2}$ such that S_i represents the start time of activity A_i. C_i denotes the completion time of activity A_i, with $C_i = S_i + p_i$. S_0 is a reference point for the start of the project. Here we assume that $S_0 = 0$. A solution S is feasible if it is compatible with the precedence constraints (1.1) and the resource constraints (1.2) expressed below, where $\mathcal{A}_t = \{A_i \in \mathcal{A} \,|S_i \leq t < S_i + p_i\}$ represents the set of non-dummy activities in process at time t.

$$S_j - S_i \geq p_i \quad \forall (A_i, A_j) \in E \tag{1.1}$$

$$\sum_{A_i \in \mathcal{A}_t} b_{ik} \leq B_k \quad \forall R_k \in \mathcal{R}, \forall t \geq 0 \tag{1.2}$$

The makespan of a schedule S is equal to S_{n+1}, the start time of the end activity. The above-defined set $\mathcal{A}_t$ and constraints state that an activity cannot be interrupted

1. We will identify in the rest of the chapter each activity with the corresponding node of the precedence graph

once it is started. This is referred to as not allowing *preemption*[2]. The RCPSP can then be stated as follows:

DEFINITION 1.1.– *The RCPSP is the problem of finding a non-preemptive schedule S of minimal makespan S_{n+1} subject to precedence constraints (1.1) and resource constraints (1.2).*

An important preliminary remark is that, since durations are integers, we can restrict ourselves to integer schedules without missing the optimal solution. A non-integer feasible schedule can be transformed into an integer feasible schedule without increasing the makespan by recursively applying the following principle. Consider a non-integer schedule S and let A_i denote the first activity in the increasing start time order such that $S_i \notin \mathbb{N}$. Then setting S_i to its nearest lower integer $\lfloor S_i \rfloor$ does not violate any precedence constraints, since the completion time of the predecessors of A_i are integers strictly lower than S_i. Left shifting an activity can violate a resource constraint only if it enters new sets $\mathcal{A}_t$ for $\lfloor S_i \rfloor \leq t < S_i$. Since we have $\mathcal{A}_t \subseteq \mathcal{A}_{S_i} \setminus \{i\}$ for $\lfloor S_i \rfloor \leq t < S_i$, no resource-constraint violation can appear by setting S_i to $\lfloor S_i \rfloor$. The set of integer schedules, containing at least one optimal solution, is said to be dominant.

1.2. A simple resource-constrained project example

In Table 1.1, an RCPSP instance is given with $n = 10$ real activities and $|\mathcal{R}| = 2$ resources with availabilities $B_1 = 7$ and $B_2 = 4$.

A_i	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
p_i	0	6	1	1	2	3	5	6	3	2	4	0
b_{i1}	0	2	1	3	2	1	2	3	1	1	1	0
b_{i2}	0	1	0	1	0	1	1	0	2	2	1	0

Table 1.1. *A project with 10 real activities and 2 resources*

The precedence constraints linking the activities $A_i \in V$ are displayed in Figure 1.1 as an activity-on-node graph. A schedule of minimal makespan $S^*_{n+1} = 12$ is displayed in Figure 1.2 as a 2-dimensional Gantt chart where the x axis represents the time and the y axis represents the resource occupation.

1.3. Computational complexity

According to the computational complexity theory [GAR 79], the RCPSP is one of the most intractable combinatorial optimization problems. Indeed, the RCPSP belongs

2. The preemptive case is presented in Chapter 8

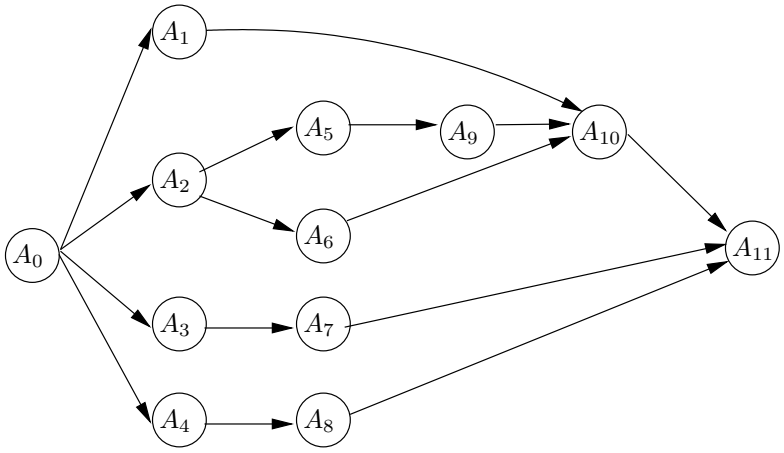


Figure 1.1. *Precedence activity-on-node graph*

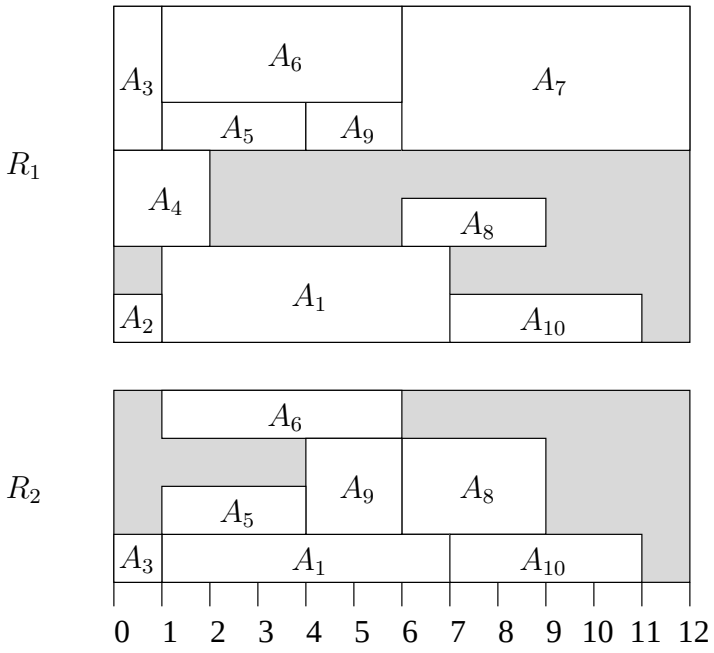


Figure 1.2. *Schedule of minimal makespan*

to the class of problems that are *NP-hard in the strong sense*. The complexity theory states that an optimization problem is NP-hard (in the strong sense) if its decision version is *NP-complete* (in the strong sense). Let us define the decision variant of the RCPSP. Let H denote an arbitrarily large integer.

DEFINITION 1.2.– *The decision variant of the RCPSP is the problem of determining whether a schedule S of makespan S_{n+1} not greater than H subject to precedence constraints (1.1) and resource constraints (1.2) exists or not.*

Roughly, a decision problem is in NP if one can verify in polynomial time if a tentative solution is feasible. Let S denote a schedule. Precedence feasibility can be trivially checked in $O(|E|)$ and the makespan constraint in $O(1)$. Verifying the resource-feasibility in polynomial time is not a trivial task. Algorithm 1 verifies the feasibility according to resource-constraints in $O(n^2|\mathcal{R}|)$. The algorithm is based on the observation that each change over time of the activity total resource requirements occurs only at time $S_0 = 0$ or at the completion time of an activity. List $\mathcal{L}$ being a sorted list of the different activity completion times, the algorithm computes each set F of activities in process at each time $t \in \mathcal{L}$, and tests whether the total resource requirements of each set F exceeds the resource availabilities. Note that directly verifying that the resource availability is respected by computing set $\mathcal{A}_t$ for each time period $t \in [0, H[$ yields a pseudo-polynomial algorithm, which depends on value H.

Algorithm 1 Resource feasibility checking of a tentative schedule S

```
1: L = {C_j | A_j ∈ V}
2: sort L in increasing values
3: for t ∈ L do
4:    for R_k ∈ R do
5:       o ← 0, F ← ∅
6:       for A_j ∈ A do
7:          if S_j < t, C_j ≥ t and b_jk > 0 then
8:             o ← o + b_jk, F ← F ∪ A_j
9:          end if
10:         if o > B_k then
11:            S is not resource-feasible because activities of F are executed in parallel
12:            return false
13:         end if
14:      end for
15:   end for
16: end for
17: return true
```

Garey and Johnson [GAR 75] have shown that the decision variant of the RCPSP with a single resource and no precedence constraints, called the resource-constrained scheduling problem, is NP-complete in the strong sense by reduction from the 3-partition problem. NP-hardness can be shown by a simpler observation made by Blazewicz *et al.* [BLA 83] yielding even worse negative results. Let us consider the famous graph coloring problem. Let $G(\mathcal{V}, \mathcal{E})$ be an undirected graph and let c be an arbitrary integer. Deciding if there is a feasible coloring of the nodes in G with c colors such that two adjacent nodes do not share the same color is NP-complete in the strong sense and appears to be a particular case of the decision variant of an RCPSP with unit durations, no precedence constraints, a disjunctive resource per arc in $\mathcal{E}$ and an activity per node in $\mathcal{V}$ with a unit requirement on each resource of its incident arcs. The coloring problem is feasible if and only if the RCPSP has a makespan not greater than c. Figure 1.3 illustrates the 3 colored solution as a RCPSP of makespan 3. Switching to optimization, the minimal number of colors needed, corresponding to the minimal makespan, is called the chromatic number of the graph.

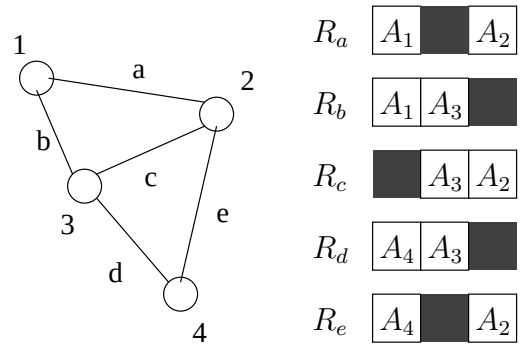


Figure 1.3. *A schedule of minimal makespan equal to the chromatic number of the displayed graph $G(\mathcal{V}, \mathcal{E})$*

When dealing with problems that are NP-hard in the strong sense, an important question is the existence of a polynomial-time approximation scheme. Uetz [UET 01] noticed that by a direct extension of non-approximability results obtained by Feige and Kilian [FEI 98] on graph coloring, it is very unlikely[3] that the minimal makespan can be approximated in polynomial time with a factor of $n^{1-\epsilon}$ for any constant $\epsilon > 0$.

1.4. Dominant and non-dominant schedule subsets

Let us consider the possibly empty set $\mathcal{S}_H$ of feasible schedules of a decision variant of the RCPSP. If we ignore the precedence constraints and considering that integer

3. We refer to [UET 01] and [FEI 98] for the precise conditions

solutions are dominant, the possible values for each start time S_i can be restricted to interval $[0, H - p_i[$ yielding possibly $\Pi_{A_i \in \mathcal{A}}[0, H - p_i[$ schedules.

Fortunately, in spite of the computational complexity of the problem stated in the previous section, it is possible to avoid the complete enumeration of this possibly huge search space to find a feasible solution. It is possible to define subsets of schedules significantly smaller than $\mathcal{S}_H$ that always contain a feasible solution if it exists, i.e. dominant subsets. These sets of schedules are the sets of semi-active and active schedules, based on the concepts of local and global left shifts, respectively. Let us consider a schedule S and an activity A_i. The global left shift operator $LS(S, A_i, \Delta)$ transforms schedule S into an identical schedule S', except for $S'_i = S_i - \Delta$ with $\Delta > 0$. A left shift is local when, in addition, all schedules obtained by $LS(S, A_i, \rho)$ with $0 < \rho \leq \Delta$ are also feasible. We define the set of semi-active and active schedules as follows.

DEFINITION 1.3.– *A schedule is semi-active if it admits no feasible activity local left shift.*

From this definition, an integer schedule is semi-active if and only if for each activity $A_i \in \mathcal{A}$, there is no feasible activity left shift of 1 time unit.

DEFINITION 1.4.– *A schedule is active if it admits no feasible activity global left shift.*

From this definition, an integer schedule is active if and only if for each activity $A_i \in \mathcal{A}$, there is no feasible left shift of $\Delta \in \mathbb{N}^*$ time units. Note that the schedule displayed in Figure 1.2 is both semi-active and active.

Any solution of $\mathcal{S}_H$ can obviously be transformed into a semi-active schedule by applying a series of local left shifts. Any semi-active schedule can in turn be transformed into an active schedule by performing a series of global left shifts. It follows that the semi-active schedule set $\mathcal{S}_H^{sa}$ and the active schedule set $\mathcal{S}_H^a$ are dominant subsets of $\mathcal{S}_H$. More precisely we have $\mathcal{S}_H^a \subseteq \mathcal{S}_H^{sa} \subseteq \mathcal{S}_H$. These sets are also dominant for the search of a solution of $\mathcal{S}$ minimizing a regular objective function $f : \mathbb{R}^{n+2} \rightarrow \mathbb{R}$. A function f is regular if for all $S, S' \in \mathbb{R}^{n+2}$ such that $S \leq S'$ (where $\leq$ is meant componentwise), we have $f(S) \leq f(S')$. In particular, the makespan criterion considered in the standard RCPSP corresponds to a regular objective function.

For practical reasons it can be relevant to consider non-dominant subsets of schedules, i.e. that may exclude the optimal solution. The non-delay scheduling concept is linked to the requirement that no resource is left idle while it could process an activity. Although it may appear intuitively as efficient, this scheduling policy may lead only to sub-optimal solutions. The set of non-delay schedules can be defined by considering partial left shifts. A partial left shift consists of allowing preemption for the

left-shifted activity. An activity is preemptive if it can be interrupted at any (not necessarily integer) time-point (see Chapter 8). Thus, a partial left shift $PLS(S, A_i, \Delta, \Gamma)$ of an originally non-interrupted activity consists of a left shift of $\Delta > 0$ time units of the first $\Gamma > 0$ units of the activity.

DEFINITION 1.5.– *A schedule is non-delay if it admits no feasible activity partial left-shift.*

For an integer schedule S, whenever a resource is left idle while it could start processing an activity, it is possible to globally left-shift the first unit of this activity into the idle period. From the above definition, the set $\mathcal{S}_H^{nd}$ of semi-active schedules of makespan not greater than H verifies $\mathcal{S}_H^{nd} \subseteq \mathcal{S}_H^{a}$. The schedule displayed in Figure 1.2 is also a non-delay schedule .

Introduced for the job-shop problem by Baker [BAK 74], similar definitions of semi-active, active and non-delay schedule sets for the RCPSP can be found in [SPR 95, NEU 00a].

1.5. Order-based representation of schedules and related dominant schedule sets

From the dominance property of the semi-active schedule set, we may derive a relevant representation of schedules. Let S be a feasible schedule. For any activity A_j, if $S_j > 0$ and if there is no other activity A_i such that $S_i + p_i = S_j$, then activity A_j can be left-shifted and the schedule is not semi-active. Hence, in any semi-active schedule S and for any activity A_j such that $S_j > 0$, there is an activity A_i such that $S_j = S_i + p_i$. It follows that it is intuitively relevant to represent a left-shifted schedule by additional precedence constraints. Bartusch *et al.* [BAR 88] propose to consider the decomposition of set $\mathcal{S}$ of feasible solutions into finitely many subsets, each one including schedules satisfying the precedence constraints induced by a different strict order. Formally, let P denote a strict order[4] on the set of activities $\mathcal{A}$ and let $\mathcal{S}(P) = \{S | S_j - S_i \geq p_i \quad \forall (A_i, A_j) \in E \cup P\}$ denote the set of (not necessarily feasible) schedules that verify the precedence constraints given by E and the ones given by P. $\mathcal{S}(P)$ is a polyhedron and so is $\mathcal{S}(\emptyset)$ which is the set of time-feasible schedules. By definition, a strict order P is time-feasible if $\mathcal{S}(P) \neq \emptyset$ and is (resource- and time-) feasible if, in addition, $\mathcal{S}(P) \subseteq \mathcal{S}$. A result established by Bartusch *et al.* [BAR 88] states that the set of schedules $\mathcal{S}$ is the union of the polyhedra $\mathcal{S}(P)$ for all inclusion-minimal feasible strict orders P on the set of activities $\mathcal{A}$. In the case of the search for a feasible solution in $\mathcal{S}_H$, or the minimization of a regular objective function in $\mathcal{S}$, the minimal element of $\mathcal{S}(P)$ dominates the other elements of this set. This is the point $ES(P)$ (earliest-start schedule) of $\mathcal{S}(P)$ such that $\forall S' \in \mathcal{S}(P)$, $ES(P) \leq S'$.

4. A strict order is an irreflexive and transitive binary relation.

By definition, $ES(P)_i$ is equal to the length of the longest path from A_0 to A_i in graph $G(V, E \cup P)$, each arc $(A_i, A_j) \in E \cup P$ being valuated by p_i. It follows that $\mathcal{S}(P) \neq \emptyset$ if and only if $G(V, E \cup P)$ is acyclic.

The RCPSP can be defined as the search for a feasible strict order P on $\mathcal{A}$ such that $ES(P)$ is of minimal makespan. Note that P is said to be feasible not only if $ES(P)$ is feasible, but if and only if all schedules in $\mathcal{S}(P)$ are feasible. We will see how this type of feasibility can be checked in the next section to specify the strict order-based RCPSP formulation.

To illustrate these concepts, let us consider schedule S displayed in Figure 1.2 and strict orders
$P_1 = \{(A_2, A_1), (A_3, A_1), (A_3, A_6), (A_6, A_7), (A_3, A_7), (A_9, A_7), (A_9, A_8)\}$,
$P_2 = P_1 \cup \{(A_3, A_8), (A_6, A_8)\}$ and $P_3 = P_2 \setminus \{A_9, A_8\}$.

Note that we have $S = ES(P_1) = ES(P_2) = ES(P_3)$. P_1 and P_2 are both feasible strict orders since any schedule in $\mathcal{S}(P_1)$ or $\mathcal{S}(P_2)$ is feasible. However, P_3 is not a feasible strict order although $ES(P_3)$ is feasible. Indeed, consider schedule $S' = S$, except that S_9 and S_7 are both right-shifted by 1 time unit. S' is in $\mathcal{S}(P_3)$ but is not resource-feasible since R_2 is oversubscribed at time C_9. As stated above, it is not trivial to check the feasibility of a strict order, which corresponds here to see that $\mathcal{S}(P_1) \subseteq \mathcal{S}$ and $\mathcal{S}(P_2) \subseteq \mathcal{S}$. This is linked to the forbidden set concept described in section 1.6. However, this example shows that the above-defined feasibility condition of a strict order is not a necessary condition for obtaining a feasible schedule. We can also remark that the *important* arcs of a feasible strict order are the ones belonging to its transitive reduction. Let us consider $P_1' = P_1 \setminus (A_3, A_7)$, the transitive reduction of P_1. We have $\mathcal{S}(P_1) = \mathcal{S}(P_1')$. Furthermore, P_1 is an inclusion-minimal feasible strict order: if we remove any arc belonging to the transitive reduction of P_1 (i.e. all arcs except (A_3, A_7)), $ES(P_1)$ becomes unfeasible. P_2 is not an inclusion-minimal feasible strict order since we have $P_2 \subset P_1$. This shows that several strict orders may lead to the same earliest-start schedule.

Another important remark is that despite the fact that the semi-active concept is intuitively related to the strict order concept, each earliest-start schedule with respect to a given strict order does not necessarily correspond to a semi-active schedule in the sense of Definition 1.3 as noted by Sprecher *et al.* [SPR 95]. For instance, let us define strict order $P_4 = P_1 \cup \{(A_{10}, A_8)\}$. The earliest start schedule $ES(P_4)$ is identical to the one displayed in Figure 1.2 except that the start time of activity 8 is delayed up to the completion time of activity A_{10}. Since a local left shift of activity A_8 is made feasible, schedule $ES(P_4)$ is not semi-active. For that reason, Neumann *et al.* [NEU 00a] propose additional schedule sets related to the strict order representation: the quasi-active and the pseudo-active schedule sets. These sets are based on the order-preserving and order-monotonic left-shifts, respectively. These left shifts are defined by reference to the interval order $P(S)$ induced by a schedule S

where $P(S) = \{(A_i, A_j) \in V^2 | S_j \geq S_i + p_i\}$. By definition, $P(S)$ is the inclusion maximal strict order P such that $S \in \mathcal{S}(P)$. Since $P(S)$ is inclusion maximal, each antichain in graph $G(V, E \cup P(S))$ represents a set of activities simultaneously in process.

An order-preserving left shift of an activity on S yields a feasible schedule S' such that $P(S) \subseteq P(S')$. An order-monotonic left-shift of an activity on S yields a feasible schedule S' such that $P(S) \subseteq P(S')$ or $P(S') \subseteq P(S)$. From this definition it follows that in an order-monotonic left shift from schedule S, either all chains of $G(V, E \cup P(S))$ or all antichains are preserved. Consequently, an order-preserving left shift is also order-monotonic. Moreover, since the initial and final schedules belong to the same feasible order polyhedron $\mathcal{S}(P(S))$ or $\mathcal{S}(P(S'))$, an order-monotonic left shift is also a local left-shift.

DEFINITION 1.6.– *A schedule is quasi-active if it admits no feasible order-preserving left shift.*

From a polyhedral point of view, a schedule S is quasi-active if for each activity $A_i \in V$, $S' = LS(S, A_i, \Delta)$ does not belong to $\mathcal{S}(P(S))$, for all $\Delta > 0$, i.e. if it is the minimal point of $\mathcal{S}(P(S))$.

We can show that schedule $ES(P_4)$ defined above is quasi-active since any left shift violates its interval order $P(ES(P_4)) = \{(A_1, A_8), (A_1, A_{10}), (A_2, A_1), (A_2, A_5), (A_2, A_6), (A_2, A_7), (A_2, A_8), (A_2, A_9), (A_2, A_{10}), (A_3, A_1), (A_3, A_5), (A_3, A_6), (A_3, A_7), (A_3, A_8), (A_3, A_9), (A_3, A_{10}), (A_4, A_7), (A_4, A_8), (A_4, A_9), (A_4, A_{10}), (A_5, A_7), (A_5, A_8), (A_5, A_9), (A_5, A_{10}), (A_6, A_7), (A_6, A_8), (A_6, A_{10}), (A_9, A_7), (A_9, A_8), (A_9, A_{10}), (A_{10}, A_8)\}$.

Since each semi-active schedule is also a quasi-active schedule, we have the following relations $\mathcal{S}_H^{sa} \subseteq \mathcal{S}_H^{qa} \subseteq \mathcal{S}_H$ where $\mathcal{S}_H^{qa}$ denote the set of quasi-active schedules.

DEFINITION 1.7.– *A schedule is pseudo-active if it admits no feasible order-monotonic left shift.*

A schedule S is pseudo-active if for each activity $A_i \in V$, $S' = LS(S, A_i, \Delta)$ does not belong to $\mathcal{S}(P)$, for all $\Delta > 0$ and for all strict orders P such that $S \in \mathcal{S}(P)$, i.e. if it is the minimal point of all schedule sets $\mathcal{S}(P)$ containing S.

Schedule $ES(P_4)$ is not pseudo-active since a local left shift of 1 unit on activity A_{10} is order-monotonic: it removes $(10, 8)$ and $(1, 8)$ from $P(ES(P_4))$ yielding a necessarily feasible and non-worse schedule. In the terms of Definition 1.7, we can also remark alternatively that $ES(P_4)$ belongs to the polyhedron $\mathcal{S}(P_1)$ and that activity A_{10} can be locally left shifted while obtaining a schedule still in set $\mathcal{S}(P_1)$.

Based on these definitions, the following theorem can be established.

THEOREM 1.1.– *The set of semi-active schedules and the set of pseudo-active schedules are identical for the RCPSP*

Proof. Since there is no feasible local left shift from a semi-active schedule, the set of semi-active schedules is included in the set of pseudo-active schedules. We show that, conversely, there is no feasible local left shift from a pseudo-active schedule. Suppose that there is a feasible local non-order-monotonic left shift $LS(S, A_i, \Delta)$ from a pseudo-active schedule S yielding schedule S'. Since $LS(S, A_i, \Delta)$ is not order-monotonic, let $(A_j, A_i) \in P(S)$, $(A_j, A_i) \notin P(S')$, $(A_i, A_k) \notin P(S)$ and $(A_i, A_k) \in P(S')$. Let $\Delta' = (S_i + p_i - S_k)/2$. We have $\Delta > \Delta'$ and then $LS(S, A_i, \Delta')$ is feasible and does not create (A_i, A_k) while (A_j, A_i) is either removed from or kept in $P(S')$. From S', we can recursively apply this principle until obtaining a feasible order-monotonic left shift from S, which contradicts the hypothesis. ■

To sum up, we have the following relations between the presented dominant schedule sets: $\mathcal{S}_H^a \subseteq \mathcal{S}_H^{sa} = \mathcal{S}_H^{pa} \subseteq \mathcal{S}_H^{qa} \subseteq \mathcal{S}_H$, where $\mathcal{S}_H^{pa}$ denotes the set of pseudo-active schedules. It should be mentioned that for the RCPSP with minimal and maximal time lags, an extension of the problem considered in Chapter 11, it generally holds that $\mathcal{S}_H^{sa} \subset \mathcal{S}_H^{pa}$ [NEU 00a].

1.6. Forbidden sets and resource flow network formulations of the RCPSP

From the order-based representation of dominant schedules, we can derive two alternative formulations of the RCPSP restricting the search space to the dominant set of quasi-active schedules.

A forbidden set is a set F of activities that cannot be scheduled in parallel in a feasible solution because of resource limitations due to a resource R_k such that $\sum_{A_i \in F} b_{ik} > B_k$. A forbidden set is minimal if any of its subsets of activities F' verifies, $\forall R_k \in \mathcal{R}$, $\sum_{A_i \in F'} b_{ik} \leq B_k$. Let $\mathcal{F}$ denote the set of minimal forbidden sets. No element of $\mathcal{F}$ can be an antichain of $G(V, E \cup P(S))$ for any feasible schedule S. Consequently, a necessary condition for feasibility of a schedule S is that $\forall F \in \mathcal{F}$, $F^2 \cap P(S) \neq \emptyset$. A forbidden set F such that $F^2 \cap P(S) \neq \emptyset$ is said to be broken up. If, in addition, $S \in \mathcal{S}(\emptyset)$, we obtain a necessary and sufficient condition. Consequently, the RCPSP can be reformulated as follows:

DEFINITION 1.8.– *The RCPSP aims at finding a strict order P such that $G(V, E \cup P)$ is acyclic, $\forall F \in \mathcal{F}$, $F^2 \cap P \neq \emptyset$ and $ES(P)_{n+1}$ is minimal.*

The drawback of the preceding formulation is that the number of minimal forbidden sets $|\mathcal{F}|$ can be extremely large. A significant number of forbidden sets are implicitly broken up due to precedence relations. Indeed as soon as there is a path

from A_i to A_j in the precedence graph $G(V, E)$, any precedence-feasible schedule breaks up any forbidden set including both A_i and A_j. Thus, we do not have to consider such forbidden sets in $\mathcal{F}$.

However, it is not trivial to efficiently enumerate all the minimal forbidden sets. Stork and Uetz propose in [STO 05] an algorithm based on the enumeration of all the subsets of $\mathcal{A}$ through a tree, in which each node, except the root node, is associated with an activity. In the tree, each branch corresponds to a candidate minimal forbidden set. The root node has n children corresponding to A_1,...,A_n. Then, each node A_i has potentially $n-i$ children A_{i+1},...,A_n. A node corresponds to a minimal forbidden set as soon as the sum of the resource demands over the node itself and all its ascendants exceeds the capacity of one resource. A node is fathomed as soon as it does not lead to a minimal forbidden set. Figure 1.4 displays the 24 minimal forbidden sets, not broken up by precedence relations, obtained through the tree structure for the illustrative example. Note that because of precedence relations, activity A_{10} does not belong to any forbidden set and, consequently, can be ignored for the computation of the strict orders. If we ignore the precedence relations, the number of minimal forbidden sets grows up to 97. The reader may verify that both strict orders P_1 and P_2 break up the 24 forbidden sets whereas P_3 does not (see for instance forbidden set $\{A_1, A_8, A_9\}$). Chapter 7 discusses computational experiments implementing the Stork and Uetz algorithm to evaluate the hardness of RCPSP instances.

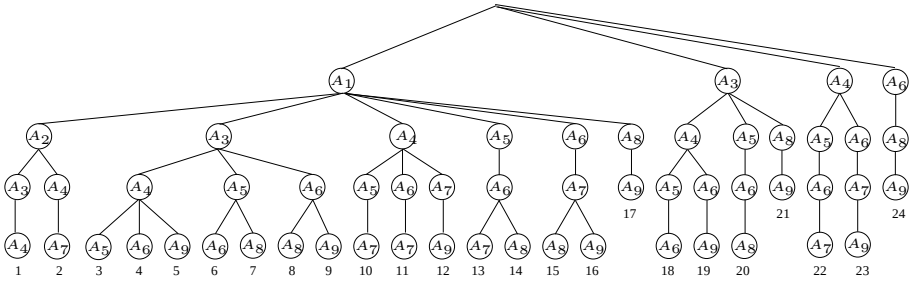


Figure 1.4. *Minimal forbidden sets of the illustrative example*

An intermediate order-based representation can be used to avoid the use of $\mathcal{F}$ in the RCPSP formulation. This representation involves the concept of resource flow decision variables f_{ij}^k for $A_i \in \mathcal{A} \cup \{A_0\}$, for $A_j \in \mathcal{A} \cup \{A_{n+1}\}$ and for $R_k \in \mathcal{R}$. f is a resource flow if it verifies the following constraints:

$$f_{ij}^k \geq 0 \quad \forall A_i \in \mathcal{A} \cup \{A_0\}, \forall A_j \in \mathcal{A} \cup \{A_{n+1}\}, \forall R_k \in \mathcal{R} \tag{1.3}$$

$$\sum_{A_i \in \mathcal{A} \cup \{A_{n+1}\}} f_{0i}^k = \sum_{A_i \in \mathcal{A} \cup \{A_0\}} f_{i(n+1)}^k = B_k \quad \forall R_k \in \mathcal{R} \tag{1.4}$$

$$\sum_{A_j \in \mathcal{A} \cup \{A_{n+1}\}} f_{ij}^k = \sum_{A_j \in \mathcal{A} \cup \{A_0\}} f_{ji}^k = b_{ik} \quad \forall A_i \in \mathcal{A}, \forall R_k \in \mathcal{R} \tag{1.5}$$

A resource flow f induces the strict order $P(f)$, the transitive closure of $\{(A_i, A_j) \in V^2 | f_{ij} > 0\}$. The following lemmas establish the link between flows and strict orders.

LEMMA 1.1.– *If f is a flow and $G(V, E \cup P(f))$ is acyclic then $P(f)$ is a feasible strict order.*

Proof. First, if $G(V, E \cup P(f))$ is acyclic, $ES(P(f))$ is time feasible. Second, suppose that a forbidden set F is not broken up by $ES(P(f))$ for a resource k. Since all activities of F are simultaneously in process during at least one time period, there is a total incoming flow of F equal to $\sum_{i \in F} b_{ik} > B_k$ which violates the flow conservation conditions. ■

Consequently, a resource flow such that $G(V, E \cup P(f))$ is acyclic is said to be feasible.

LEMMA 1.2.– *For each feasible strict order P, there is a feasible resource flow f such that $ES(P(f)) \leq ES(P)$.*

Proof. Let us consider the feasible schedule $S = ES(P)$ and strict order $P(S) \supseteq P$ with $ES(P(S)) = S$. By setting additional constraints $f_{ij}^k = 0$ for all $(A_i, A_j) \notin P(S)$ we show that there exists a flow verifying equations (1.3)-(1.5). This amounts to verifying that there exists a feasible flow in each network $G_k(V, E \cup P(S))$, for each resource $R_k \in \mathcal{R}$ with minimal and maximal node capacities b_{ik} for all activities $A_i \in \mathcal{A}$ and node capacities B_k for nodes A_0 and A_{n+1}. Let us transform this graph with bounds on node flow into a graph with bounds on arc flows. We split each node $A_i \in V$ into two nodes A_i and A_i' linked by an arc of minimal and maximal capacities equal to the node capacity. All other arcs have null minimal capacities and infinite maximal capacities. If we now ignore the maximal capacities, since $P(S)$ is feasible, there is no $A_0 - A_{n+1}'$ cut of (minimal) capacity greater than B_k. Indeed suppose that such a cut exists. Then, it includes a set of arcs (i, i') that represents a non-broken up forbidden set. Furthermore there is a $A_0 - A_{n+1}'$ cut (reduced to arc (A_0, A_0')) of minimal capacity equal to B_k. Therefore, according to the min-flow max-cut theorem (see [NEU 03, LEU 04]), we have a minimal flow equal to B_k. ■

It follows from Lemmas 1.1 and 1.2 that the RCPSP can be defined as follows.

DEFINITION 1.9.– *The RCPSP aims at finding a feasible flow f such that $ES(P(f))_{(n+1)}$ is minimal.*

From a feasible flow f, a feasible schedule $ES(P(f))$ can be computed by longest path computations in graph $G(V, E \cup P(f))$ where all arcs $(A_i, A_j) \in E \cup P(f)$ are valuated by p_j.

Conversely, Algorithm 2 allows the computation of a feasible flow f from any feasible schedule S in $O(n^2|\mathcal{R}|)$. The algorithm assumes activities are sorted in increasing order of their completion times. For each activity A_i, the incoming flow is simply taken from the activities A_j completed before the start time of A_i. At each step of the algorithm, β_{ik} denotes the number of resource R_k units that remain to be sent to A_i.

Algorithm 2 BUILDFLOWFROMSCHEDULE(f, S): generates flow f from schedule S

$\beta_{ik} \leftarrow b_{ik}, \forall A_i \in \mathcal{A}, \forall R_k \in \mathcal{R}$
$f^k_{i(n+1)} \leftarrow b_{ik}, \forall A_i \in \mathcal{A}, \forall R_k \in \mathcal{R}$
$f^k_{0(n+1)} \leftarrow B_k, \forall R_k \in \mathcal{R}$
for $A_i \in \mathcal{A}$ (in increasing order of completion times C_i) **do**
  for $A_j \in \mathcal{A}$, $A_j \neq A_i$ **do**
    for $R_k \in \mathcal{R}$ **do**
      if $C_j \leq S_i$ **then**
        $\rho \leftarrow \min(f^k_{j(n+1)}, \beta_{ik})$
        $\beta_{ik} \leftarrow \beta_{ik} - \rho$
        $f^k_{j(n+1)} \leftarrow f^k_{i(n+1)} - \rho$
        $f^k_{ji} \leftarrow f^k_{ji} + \rho$
      end if
    end for
  end for
end for

Figure 1.5 gives a flow representing the schedule displayed in Figure 1.2. The flow values for both resources are displayed near each arc. Dotted arcs are the arcs representing the additional precedence constraints induced by the flow and used to define $P(f)$. We observe here that $P_1 \subset P(f)$, P_1 being the inclusion-minimal strict order defined in section 1.5. Consequently, one can verify on this example that $P(f)$ breaks up all the forbidden sets.

1.7. A simple method for enumerating a dominant set of quasi-active schedules

A very simple algorithm can be designed to find an optimal solution of a RCPSP without enumerating the forbidden sets nor using the concept of flows. Conceptually, it consists of starting with an empty strict order $P = \emptyset$. If the time feasible earliest-start schedule $ES(P)$ is feasible, which can be checked by Algorithm 1, then it is the

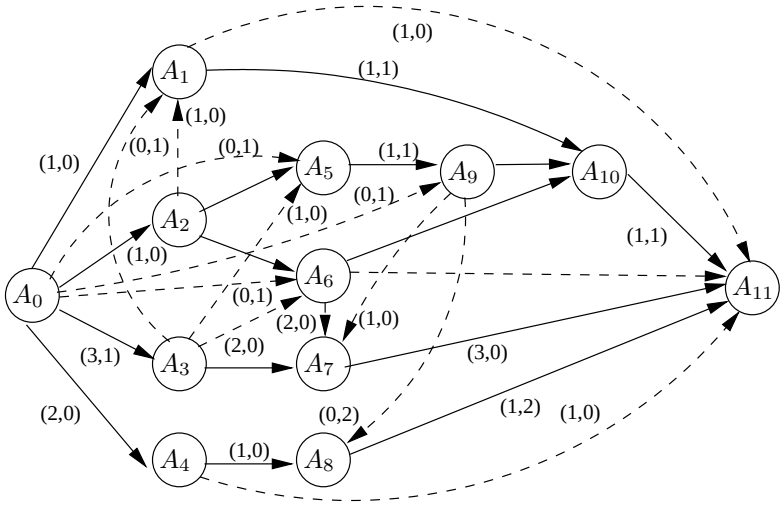


Figure 1.5. *A resource flow for the schedule of Figure 1.2*

optimal schedule. Otherwise, Algorithm 1 returns a non-broken up minimal forbidden set F. The optimal solution can be found by recursively applying this procedure for each $(A_i, A_j) \in F$ after adding (A_i, A_j) to P. The recursive algorithm OPTFS is given as Algorithm 3.

Algorithm 3 OPTFS(P): search for an optimal solution to the RCPCP with an initial set of additional arcs P

1: **if** $\mathcal{S}(P) = \emptyset$ **then**
2: return $+\infty$
3: **else if** $ES(P)$ is resource-feasible (*checked by Algorithm 1)* **then**
4: return $ES(P)_{n+1}$
5: **else**
6: (*Algorithm 1 has returned a violated forbidden set* F*)*
7: return $\min_{(A_i,A_j)\in F\times F}$OPTFS$(P \cup \{(A_i, A_j)\})$
8: **end if**

Algorithm 3 can also be used as a one pass method to build a single feasible solution to the RCPSP. It suffices to select at each step a single arc $(A_i, A_j) \in F^2$ such that (A_i, A_j) does not induce a cycle in $G(V, E \cup P)$. An $O(|E \cup P|)$ algorithm can be used to perform a cycle check. Such an arc always exists because if $(A_i, A_j) \in F^2$ induces a cycle, (A_j, A_i) does not induce a cycle and also breaks up the forbidden set. Since an arc is added at each iteration, the algorithms gives a feasible solution in $n(n-1)/2$ iterations. The obtained schedule is quasi-active since it is equal to

$ES(P)$. It follows that Algorithm 3 computes a set of dominant quasi-active schedules. In the algorithm we refer to P as a set of additional arcs and not as a strict order for practical reasons. As stated earlier, the relevant elements of a strict order are the arcs belonging to the transitive closure. When adding arc (A_i, A_j) to P at step 7, computing the transitive closure of P to keep the transitivity property would be an unnecessary computational effort.

Chapter 2

Resource and Precedence Constraint Relaxation

RCPSP is one of the most complex scheduling problems. It can be considered as a generalization of many standard scheduling subproblems. Relaxing an initial RCPSP instance in order to build relaxed subproblems may be useful for the resolution of the initial instance. Relaxations are mainly used to compute efficient lower bounds of the project duration, but also to totally or partially build a solution of the initial instance. For instance if we refer to the disjunctive problem relaxations (see section 2.2), specific branching schemes have been designed to schedule machines. Directly inspired by the large amount of work dedicated to job-shop scheduling , these branching schemes consist of ordering all activities that require the same single machine for solving the RCPSP. Ordering the single machines before applying any specifically designed branching scheme for solving RCPSP drastically improves the performance of the branch-and-bound methods. Thus, building relevant single machine scheduling instances, or parallel machine scheduling instance from a RCPSP instance may be interesting to speed up the resolution and/or to get efficient lower bounds.

In this chapter, we present traditional critical path based lower bounds and methods for bounding single resource problems that arise as relaxations of the RCPSP. Basically, these new problems are obtained either by relaxing the precedence relations to time-windows for activities and/or by relaxing partially the resource constraints. Namely these problems are (1) the single machine problem ($1|r_i, q_i|C_{max}$), (2) the identical parallel machine problem ($P|r_i, q_i|C_{max}$) and (3) the multiprocessor problem ($P|r_i, q_i, size_i|C_{max}$) also known as the Cumulative Scheduling Problem

Chapter written by Emmanuel NÉRON.

(CuSP). For these three relaxations, we first describe how they can be derived from an initial RCPSP instance, and we present briefly how they can be used to compute lower bounds for RCPSP instance.

2.1. Relaxing resource constraints

The most intuitive relaxation of the RCPSP consists of relaxing all the resource constraints. Then the corresponding problem is to find a critical path in a directed and acyclic graph that can be solved using, for instance, Bellman's algorithm. This notion of length in the precedence graph is also used to compute both earliest starting time (ES_i) and tail (q_i) of activities. Tail, q_i, corresponds to the minimum duration between the end of the activity A_i and the end of the project. $ES_i = \mathcal{L}(A_0, A_i)$ and $q_i = \mathcal{L}(A_i, A_{n+1}) - p_i$, where $\mathcal{L}(A_i, A_j)$ denotes the longest path in the precedence graph between A_i and A_j. This trivial relaxation, that consists of ignoring resource constraints to compute time-bounds for activities, plays an important role in the solving methods (lower bound computation, adjusting time-windows of activities etc.). An RCPSP instance and a corresponding critical path are presented in the top part of Figure 2.1.

Some improvements have been proposed in order to enforce critical path relaxation [STI 78, DEM 92b]. Authors consider an activity that cannot be processed simultaneously (due to resource constraints) with activities of the critical path (CP). Let us consider an activity A_i with a time-window $[ES_i, LC_i]$, and let us consider that activities of the critical path are scheduled at their earliest starting times. Then, if the amount of resources required by A_i are available only during e_i time-units between ES_i and LC_i, with $e_i \leq p_i$, the project completion will be delayed by at least $p_i - e_i$ time-units, thus $L = max_{A_i \notin CP}(\mathcal{L}(A_0, A_{n+1}) + p_i - e_i)$ is a valid lower bound of the project duration. Figure 2.1 shows the Gantt diagram associated to the activities of the critical path. Notice that resource is not available to process activity A_5 simultaneously with the ones of the critical path. Thus, a new lower bound can be calculated which is equal to 12.

2.2. Calculating the disjunctive subproblem

In this section we present a method to build such single machine problems from an initial RCPSP and we quickly present an overview of techniques either to compute lower bounds of the makespan ($\max_i(C_i + q_i)$, where C_i denotes the completion time of the activity A_i) or to derive satisfiability tests, and time-window adjustments of activities.

In the single machine problem, a set of activities has to be processed on a single machine. Each activity, A_i has a release date ES_i, a processing time p_i, a tail q_i or

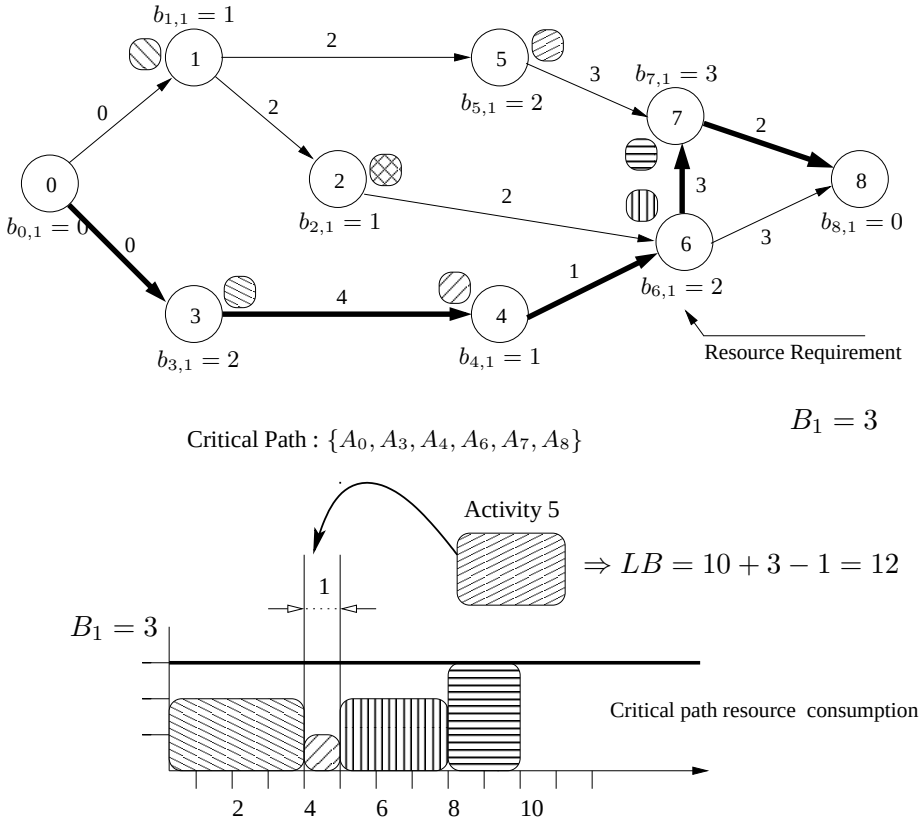


Figure 2.1. *RCPSP instance: critical path lower bound and improved critical path lower bound*

a deadline LC_i if the decision variant is considered. This problem is known to be $\mathcal{NP}$-hard in the strong sense. It plays a central role for solving the RCPSP as (1) we can derive release dates and tails of activities from the precedence constraints and (2) we can compute sets of activities that, due to machine or to precedence constraints, cannot be processed in parallel and can thus be "put" on a fictious single machine. Main notations used to formally define a one-machine problem instance are recalled in Table 2.1.

To generate redundant single machines, we look for sets of activities that are known not to overlap in any feasible solution. Note that two activities A_i and A_j never overlap in time (1) if there is a precedence constraint between A_i and A_j or (2) if there is a resource such that the total amount of resource required by A_i and A_j on the considered resource exceeds its capacity. In the following, two activities

Notation	Definition
$\mathcal{A}$	set of activity
n	number of activities: $\|\mathcal{A}\|$
p_i	processing time of activity A_i
ES_i	earliest starting time of activity A_i (release date)
q_i	tail of activity A_i
(or LC_i	latest completion time of activity A_i (deadline))

Table 2.1. *Notation for the one machine problem*

meeting conditions (1) and/or (2) are said to be "compatible". Any set of activities in which all activities are pairwise compatible is a candidate redundant machine.

One way to build redundant machines is to associate a binary variable $\Upsilon_i \in \{0, 1\}$ to each activity A_i (Υ_i equals 1 when A_i belongs to the single machine under construction, 0 otherwise). A vector Υ corresponds to a valid redundant machine if for all activities A_i, A_j that are not compatible, $\Upsilon_i + \Upsilon_j \leq 1$.

Since the constraint propagation techniques that are used either to compute lower bounds and/or deduced relevant precedence relations between activities are costly in terms of CPU time, only some redundant machines can be generated. Thus, they may be selected for instance using the load of the machines. This corresponds to finding a vector Υ that maximizes $\sum p_i \cdot \Upsilon_i$. The resulting problem is a MIP with n variables and at most n^2 constraints. In [MIN 98, BAP 00], a greedy heuristic is used to build a solution to a similar MIP. Initial experiments have shown that, in terms of final reduction of time-windows, it is much better to solve the MIP to optimality. More precisely, authors build one global redundant machine according to the above MIP and one redundant machine per initial RCPSP resource. For each cumulative resource, a redundant machine is created in which all the activities requiring more than half of the resource capacity are put (such activities never overlap in time).

Very efficient methods have been proposed either for bounding and adjusting time-bounds of activities for the one-machine problem. We briefly present one of the most efficient method: *edge-finding*. Now, we consider the decision variant of the single machine problem as defined earlier. So, we have time-windows $[ES_i, LC_i]$ (release date /deadline) in which activities have to be processed.

Edge-finding and time-bound adjustments [CAR 89] consist of deducing that some activities from a given set Ω must, can or cannot be executed first (or last) in Ω. Such deductions lead to new ordering relations ("edges" in the graph representing the partial orderings of activities) and new time bounds, i.e., strengthened release dates and deadlines. These methods of edge-finding and time-bound adjustments are

closely related to recent works using constraint programming. A formal description of these methods in the disjunctive case and more generally in the cumulative case are presented in sections 4.2.3 and 4.2.4.

Notice that some of these methods are based on the preemptive relaxation of the one-machine problem: $1|r_i, q_i|C_{max}$ is polynomial and can be solved using Jackson Preemptive Schedule that runs in $O(nlogn)$ [CAR 82]. The main idea of this algorithm is that at each relevant time point, corresponding either to a release date or a completion time of an activity, the activity processed is the one with the greatest tail (q_i) among those available. This activity is processed until it is completed or until another activity can be scheduled (a release dates). Applied to one-machine problems deduced from an initial RCPSP, this method can be used to quickly compute lower bound for RCPSP.

2.3. Deducing identical parallel machine problems

One simple way to extend single machine based lower bounds for the RCPSP is to consider that more than one activity can be in process at a given time point. Several lower bounds for the identical parallel machine problem with release dates and tails (or deadlines in the associated decision variant) have been proposed in other works. Some of these bounds are presented at the end of this section.

In the identical parallel machine problem, a set $\mathcal{A}$ of activities has to be processed on m identical parallel machines. Each activity, $A_j \in \mathcal{A}$, has a release date ES_i, a processing time p_i and a tail q_i or a deadline LC_i, and it requires one machine to be processed. These problems are known to be $\mathcal{NP}$-hard in the strong sense. Preemptive or semi-preemptive relaxations can be solved efficiently and then can be used to compute lower bounds for the RCPSP. Thus, building relevant identical parallel machine problems, considering that release dates and tails of activities are deduced from the precedence constraints of the initial RCPSP, is useful for bounding RCPSP. Main notations used to formally define an identical parallel machine problem instance are presented in Table 2.2.

To build such m-machine problems, let us consider a resource k and a set $\mathcal{J} \subseteq \mathcal{A}$ of activities of the initial RCPSP such that $|\mathcal{J}| = m + 1$ and $\sum_{A_i \in \mathcal{J}} b_{i,k} > B_k$, then at most m activities of $\mathcal{J}$ can be in process at the same time. Thus, $\mathcal{J}$ defines a m-machine problem, and the optimal makespan of this identical parallel machine problem is a lower bound of the makespan of the initial RCPSP.

In [CAR 91], considering a resource R_k, m-machine problems are built such that:

– any activity A_i satisfying $\left\lceil \frac{(m+1)b_{i,k}}{B_k} \right\rceil - 1 \geq \alpha_i$, ($\alpha_i \in \mathbb{N}$), is replaced by α_i identical activities in $\mathcal{J}$,

Notation	Definition
$\mathcal{A}$	set of activities
n	number of activities: $\|\mathcal{A}\|$
m	number of available machines
p_i	processing time of activity A_i
ES_i	earliest starting time of activity A_i (release date)
q_i	tail of activity A_i
(or LC_i	latest completion time of activity A_i (deadline))

Table 2.2. *Notation for identical parallel machine problem*

– activities such that $\sum_{i=1}^{m} b_{[i],k} > B_k$, (where $b_{[i],k}$ the $i-th$ smallest resource requirement of the activities of $\mathcal{J}$) are added to $\mathcal{J}$.

These m-machine problem instances may be used, for instance, for bounding the initial RCPSP instance. Any lower bound of the completion time ($max_{A_i \in \mathcal{A}} C_i + q_i$) of the m-machine problem instances that are built applying the previous rule is a lower bound of the makespan of the project duration of the RCPSP instance. Moreover, any time-bound adjustments computed on these m-machine instances are valid for the initial RCPSP instance.

We briefly present specific lower bounds for the m-machine problem that can be used in this context:

– *set-bound*. In [CAR 91] the authors uses into a branch-and-bound method for solving RCPSP, the set-bound applied to generated m-machine instances:

$$SB(\mathcal{J}) = \frac{1}{m}(ES_{[1]} + \ldots + ES_{[m]} + \sum_{A_i \in \mathcal{J}} p_i + q_{[1]} + \ldots + q_{[m]})$$

with $ES_{[k]}$ (respectively $q_{[k]}$) the k-smallest release date (respectively tails).

– *Preemptive and semi-preemptive relaxations for identical parallel machine problem*. It is well known that preemptive relaxation of the m machine problem is polynomially solvable and that an optimal solution can be found using a max-flow formulation [HOR 74]: there exists a flow equal to $\sum_{A_i \in \mathcal{A}} p_i$, in graph G (see Figure 2.3) if and only if there exists a preemptive feasible schedule. The computation of this satisfiability test can be done in $O(n^3)$ time.

In a recent paper Haouari and Gharbi [HAO 03] proposed to build a *semi-preemptive schedule*. They proved that if an activity $A_i \in \mathcal{A}$ verifies $LC_i - ES_i < 2 \cdot p_i$, then one machine processes activity A_i during at least $[LC_i - p_i, ES_i + p_i]$ time-units. This constraint can be simply taken into account by adding minimum flow constraints on arcs between those activities and corresponding time-intervals, once

relevant time-points ($LC_i - p_i, ES_i + p_i, \forall A_i \in \mathcal{A}$ s.t. $LC_i - ES_i < 2 \cdot p_i$) have been added. Finally, Tercinet *et al.* [TER 04] have proved that the semi-preemptive relaxation can be replaced by mixing energetic reasoning [BAP 99] (also called energy reasoning) and preemptive relaxation. The approach is quite similar to the one proposed by Haouari and Gharbi but the mandatory part of the activity that must be processed during a time-interval (also called work of the activity on the time-interval), are used as minimum capacities on the edges between activities and corresponding time-intervals.

– *Jackson Pseudo Preemptive Schedule (JPPS).* Carlier and Pinson. [CAR 98] present a lower bound for the identical parallel machine problem, called Jackson Pseudo Preemptive Schedule (JPPS). The preemption of any activity is allowed, authors also assume that a machine can be shared by several activities and that an activity can be processed on several machines at a time. Furthermore, Carlier and Pinson [CAR 04] propose adjustments of release dates for this problem, that will not be retranscribed in this chapter.
In order to build JPPS, a list scheduling algorithm is used. Its priority dispatching rule at time t is the dynamic complete tail $c_i(t) = q_i + \xi_i(t)$, where $\xi_i(t)$ is the remaining processing time of activity A_i at time t in the current schedule.

We now present an example to illustrate this approach. Let us consider a RCPSP instance made up of 8 activities, and 1 resource R_1. Precedence relationship, processing times of activities and resource requirements are presented in Figure 2.3. Let us assume that the resource capacity (B_1) is equal to 5. If we compute two trivial lower bounds for this instance that are the critical path length (6) and the average workload of the resource ($\sum_{A_i \in \mathcal{A}} b_{i,1} \cdot p_i / B_1$), we obtain a lower bound equal to $max(6, (8 + 10 + 12 + 10)/5) = 8$.

Table 2.3 presents an example of *m*-machine problem instance deduced from initial resource demands ($b_{i,1}$) and resource capacity ($B_1 = 5$). Line $\alpha_i(m = 2)$ presents the number of activities that are duplicated from activity A_i to build a 2-machine problem instance.

A_i	1	2	3	4	5	6
$b_{i,1}$	2	0	0	3	2	4
α_i (m=2)	1	0	0	1	1	2

Table 2.3. *Example of generation of a 2-machine problem instance*

Figure 2.3 presents the graph that is built in order to determine if there exists a preemptive solution of the 2-machine problem instance with a makespan lower than or

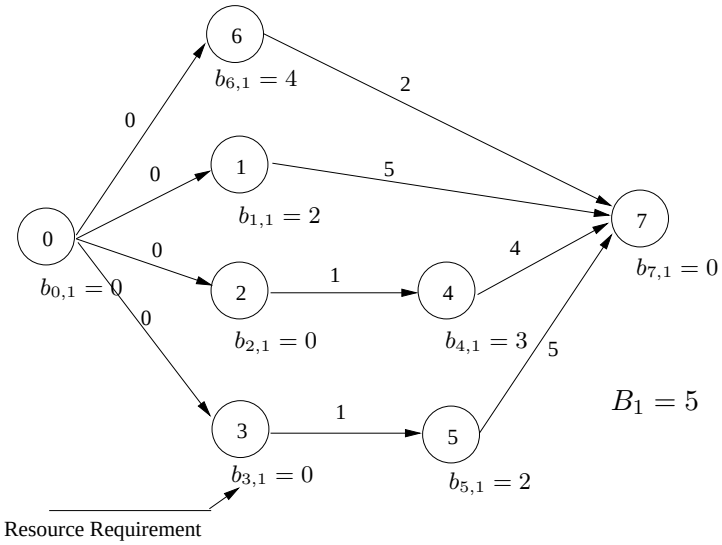


Figure 2.2. *RCPSP instance: precedence graph and resource requirements*

equal to 8. We briefly present how this graph is built. First, release dates and deadlines are computing for each activity according to the global deadline that is tested (equal to 8 in this example) and precedence relationship. Then, time-horizon is decomposed into successive time-intervals whose bounds are either release dates and/or deadlines. One node is associated with each of these time-intervals. A node is added for each activity. An arc is added from the source node to activity node, with a capacity equal to the processing time of the activity. An arc is added between an activity-node, to an interval node if the time-window of the activity overlaps the considered time-interval. The capacity of such an arc is the size of the time-interval. Finally, an arc is added between each time-interval node and the sink node, with a capacity equal to m times the length of the time-interval.

Notice that node $(6')$ and $(6'')$ correspond to the duplication of activity A_6 due to the fact that $\alpha_6 = 2$. The maximum flow in this graph is strictly lower than the sum of the processing times, then there is no preemptive solution, and we are able to conclude that 9 is a valid lower bound of the initial project duration.

Moreover, Figure 2.3 presents a preemptive schedule with a makespan equal to 9, thus 9 is the best lower bound we are able to find based on this approach using both 2-machine problem generation and preemptive relaxation.

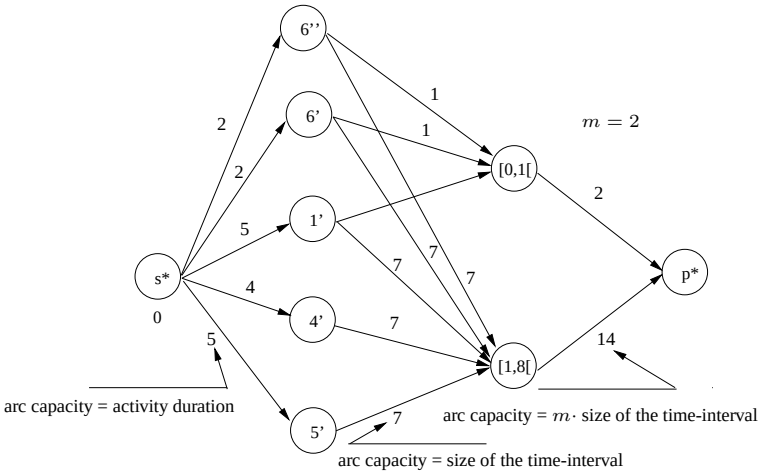


Figure 2.3. *Max-flow formulation for testing preemptive schedule*

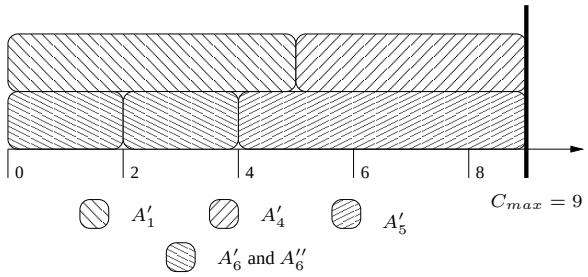


Figure 2.4. *Preemptive schedule for generated m-machine problem*

2.4. Single cumulative resource problem

This section is dedicated to techniques used to build redundant Cumulative Scheduling Problem (CuSP also denoted $P|r_i, q_i, size_i|C_{max}$) from an initial RCPSP [CAR 03, CAR 07]. These methods are based on the notion of linear lower bound and redundant resources. Lower bounds for CuSP are not presented here, but existing lower bounds proposed for RCPSP can be applied to this problem such as the energetic reasoning and its extension [BAP 99]. These methods and their recent developments are presented in Chapter 4 and more precisely in section 4.2.5.

The method that we present is based on a sub-problem where release dates and deadlines are not taken into account: we focus on specific linear forms that are lower

bounds of the makespan when a given part of an activity A_i has to be processed. These linear forms are used to obtain *redundant* CuSP, that can be used to compute efficient lower bounds for the initial instance.

The CuSP is made up of a set $\mathcal{A}$ of activities that has to be processed on a single renewable cumulative resource of capacity B. Each activity, $A_i \in \mathcal{A}$, has a release date ES_i, a processing time p_i and a deadline LC_i, and it requires b_i units of the resource to be processed. These problems are known to be $\mathcal{NP}$-Hard in the strong sense. Obtaining a relevant CuSP, considering that release dates and tails of activities are deduced from the precedence constraints of the initial RCPSP, is useful for bounding RCPSP: the CuSP can be seen as an extension of the identical parallel machine problem where activities need more than one machine to be processed. The main notation used to formally define a CuSP instance are presented in Table 2.4.

Notation	Definition
$\mathcal{A}$	set of activities
n	number of activities: $\|\mathcal{A}\|$
p_i	processing time of activity A_i
b_i	amount of resource required by activity A_i
B	resource capacity
ES_i	earliest starting time of activity A_i (release date)
LC_i	latest completion time of activity A_i (deadline)

Table 2.4. *Notation for CuSP*

Recently, the notion of *redundant functions* and *maximal redundant functions* (MRFs), and some fundamental properties of these MRF have been presented [CAR 07]. We now present the notion of Redundant Resources that is used to get relevant one-resource instances from an initial RCPSP instance restricted to one of its resources and one Redundant Function. Let us recall that b_i is the resource requirement of activity A_i, and B the resource capacity.

A redundant function (RF) f is a discrete mapping of $\{0, \ldots, B\} \rightarrow \{0, \ldots, B'\}$ ($B \in \mathbb{N}, B' \in \mathbb{N}$) such that: $i_1 + i_2 + \ldots + i_k \leq B \Rightarrow f(i_1) + f(i_2) + \ldots + f(i_k) \leq B'$. Let f be an RF $\{0, \ldots, B\} \rightarrow \{0, \ldots, B'\}$. The associated redundant resource has a capacity of B'. Moreover, activity A_i needs $f(b_i)$ units of the redundant resource.

These functions are linked to the dual feasible solution introduced in [JOH 74] for the bin-packing problem and used in [FEK 98], and [MAR 90].

Let us consider a basic lower bound for the duration of a cumulative scheduling problem, that is, the ratio of the workload divided by the available resource capacity ($\sum_{A_i \in \mathcal{A}} p_i \cdot b_i / B$). Thus, the basic bound on this redundant resource is now equal to $\sum_{A_i \in \mathcal{A}} \frac{f(b_i)}{B'} \cdot p_i$. This new bound can be larger than the one computed on the initial resource, which confirms the interest of Redundant Resources.

Unfortunately, there are Many RFs for each pair (B, B'). To restrict the search to the most relevant ones, authors introduce the notion of *Maximal Redundant Function*: a Maximal Redundant Function (MRF) is an RF such that there exists no RF f' with $f' > f$, i.e., $\forall i \in \{0, \ldots, B\}, f'(i) \geq f(i) \Rightarrow f' = f$.

Both an enumeration scheme to compute all MRF for fixed (B, B') and a linear programming formulation to detect the ones that are dominated, have been proposed [CAR 03, CAR 07]. The notion of non-dominated MRFs corresponds to MRFs that may lead to interesting lower bounds. The fact that f is a non-dominated MRF corresponds to the existence of a set of processing times of activities such that the best lower bound for these processing times is given by the redundant resource corresponding to f.

The example presented below illustrates how MRF can be used to compute lower bounds on the instance presented in Figure 2.3. Table 2.4 presents two MRFs, and the resource requirements of the activities for the redundant resources which have a capacity equal respectively to 1 and 2.

$A_j \in \mathcal{A}$	1	2	3	4	5	6
$b_{i,1}$	2	0	0	3	2	4
$f_1 : \{0, \ldots, 5\} \rightarrow \{0, 1\}$, $f_1(b_{i,1})$	0	0	0	1	0	1
$f_2 : \{0, \ldots, 5\} \rightarrow \{0, 1, 2\}$, $f_2(b_{i,1})$	1	0	0	1	1	2

Table 2.5. *An example of redundant resources*

Figure 2.4 presents an optimal schedule for the redundant resource based on f_2. Thus, we are able to conclude that 11 is a valid lower bound of the initial project duration.

2.5. Conclusion and perspectives

In this chapter we have presented how standard scheduling subproblems, namely the one-machine problem , m-machine problem and CuSP, can be deduced from an initial RCPSP instance. These subproblems are mainly used for computing lower

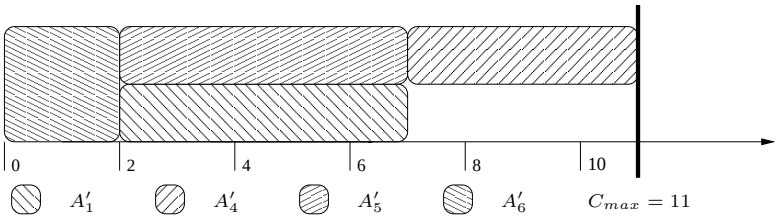


Figure 2.5. *Optimal schedule for redundant resource*

bounds that are an interesting trade-off between efficiency and computation time. Moreover, recent works have used these subproblems more explicitly, for instance, selecting disjunctions of deduced one-machine problem in order to speed up the RCPSP resolution. One interesting issue is the use of these subproblem relaxations in heuristic methods such as a large neighborhood search: solving exactly and efficiently several subproblem instances in order to build a global solution for large size RCPSP instances.

Another research direction is the identification of interesting redundant resources for a given RCPSP instance, i.e., the ones that may lead to tight lower bounds. Moreover, these relevant redundant resources may be different during the time-horizon of the project, thus introducing redundant resources related to one given time-interval may be an interesting research direction. Finally, since redundant resource focuses on each initial resource of the initial RCPSP instance separately, then a promising research direction consists of finding redundant resources that takes as accurately as possible, at least two resources of the initial RCPSP instance into account.

Chapter 3

Mathematical Programming Formulations and Lower Bounds

Most of the earliest studies [PRI 69, BAL 70, FIS 73, PAT 76, TAL 78, STI 78] devoted to the RCPSP relate to mathematical programming. Standard branch-and-bound approaches, using a relaxation of a mathematical programming formulation for the computation of lower bounds, were proposed to solve the RCPSP at optimality. However, modeling the cumulative resource constraints within a mathematical program requires dealing with a large number of variables or inequalities. Therefore, solving the linear programming relaxation of any model for the RCPSP is too time consuming – or just impossible – in practice. Branch-and-bound were implemented using weaker combinatorial relaxations (they are presented in Chapter 2) and yielded to an optimal solution for only small size instances. Some research was carried out since, to optimally solve problems of larger sizes by exploiting less drastic relaxations. More sophisticated formulations together with suited solution techniques (column generation, lagrangean relaxation) were proposed, as well as a number of specific valid linear inequalities to tighten the relaxations. Thus, the best lower bounds to date combine column generation and cutting planes with constraint programming-based preprocessing under a destructive approach. Even if, for medium size instances, such bounds require too much time to be computed at each node of a branch-and-bound, they may be particularly helpful to evaluate the quality of practical solution approaches such as metaheuristics for the RCPSP. Furthermore, the studies of these bounds have greatly contributed to a better understanding of the RCPSP and, also, on the development of combined solution techniques.

Chapter written by Sophie DEMASSEY.

In this chapter, we present the main mathematical programming formulations (either binary or mixed integer linear programs) and their relaxations proposed for the basic RCPSP defined in Chapter 1.

These formulations consist of two main classes regarding the decision variables: *sequence-based formulations* focus on the ordering the activities are processed in, one compared to another; and *time-indexed formulations* focus on the assignment of resources to activities at each time. Sequence-based models are presented in section 3.1 and time-indexed models are presented in section 3.2.

The mathematical programming approaches for the RCPSP reviewed in this chapter join up the numerous studies that have been published on the application of polyhedral approaches to scheduling problems. Most studies for the RCPSP derive from previous studies on shop scheduling with disjunctive resources. We refer to the report by Queyanne and Schultz [QUE 94] for a comprehensive synthesis of these approaches.

In order to improve the readibility of the formulations and to conform to a common usage in mathematical programming, the notations are simplified in this chapter: activities A_i, A_j and resources R_k are denoted by their indices i, j and k.

3.1. Sequence-based models

Scheduling activities on a resource can be viewed as, first, determining a sequence of the activities satisfying the precedence constraints, and then fixing the processing times of the activities following the sequence and according to other constraints such as resource availability, set-up times, release and due dates, etc. The sequence-based mathematical programming formulations reflect this decomposition. They lie on two sets of decision variables: the boolean *linear ordering* variables to determine the relative ordering of each pair of activities, and the numerical *natural date* variables to determine the processing, e.g. the start or the completion time, of each activity.

Formally, linear ordering variables $(x_{ij})_{(i,j)\in V\times V}$ are defined for each pair of activities. Variable x_{ij} takes value 1 if activity j starts after the completion of activity i, and 0 otherwise. They are linked with the starting time variables $(S_i)_{i\in V}$ as follows:

– i precedes j: $x_{ij} = 1 \wedge x_{ji} = 0 \iff S_j \geq S_i + p_i$

– j precedes i: $x_{ij} = 0 \wedge x_{ji} = 1 \iff S_i \geq S_j + p_j$

– i and j are in parallel: $x_{ij} = 0 \wedge x_{ji} = 0 \iff S_i - p_j < S_j < S_i + p_i$

– $x_{ij} = 1 \wedge x_{ji} = 1$ is not allowed (no cycle).

Sequence-based models are often used in the context of shop scheduling with disjunctive resources. For a problem with a single disjunctive machine and no additional temporal constraints, a minimum makespan schedule is uniquely determined by the

sequence of the activities, then by the instantiation of the linear ordering variables. In the cumulative case, natural date variables are also needed to characterize the schedule. Big-M constraints are then required to link the date variables to the ordering variables. This notoriously gives poor linear relaxations, but several techniques, such as cutting plane generation methods based on the pioneering works on disjunctive programming by Balas [BAL 79], may be applied to tighten these relaxations.

Essentially, two sequence-based models have been proposed for the RCPSP, formulating the resource constraints in two different ways. The formulation by Alvarez-Valdés and Tamarit [ALV 93] (section 3.1.1) is a natural extension of the linear ordering approach for disjunctive scheduling, while the formulation by Artigues [ART 03] (section 3.1.2) is based on the notion of resource flow.

3.1.1. *Minimal forbidden sets*

On disjunctive resources, the set of activities is totally ordered, hence the resource constraints can be written as $x_{ij}+x_{ji} \geq 1, \forall (i,j) \in V \times V$. On cumulative resources, as described in section 1.6, we have to delay at least one activity in any *forbidden set* $F \in \Phi$: a set of activities whose total consumption exceeds the capacity on at least one resource. The resource constraints then become $\sum_{(i,j)\in F^2} x_{ij} \geq 1, \forall F \in \Phi$.

The linear formulation proposed by Alvarez-Valdés and Tamarit ([ALV 93]) is based on this notion of forbidden sets:

$$\min \quad S_{n+1} \tag{3.1}$$

$$\text{s.t.} \quad x_{ij} = 1 \qquad \forall\, (i,j) \in E \tag{3.2}$$

$$x_{ij} + x_{ji} \leq 1 \qquad \forall\, (i,j) \in V^2, i < j \tag{3.3}$$

$$x_{ik} \geq x_{ij} + x_{jk} - 1 \qquad \forall\, (i,j,k) \in V^3 \tag{3.4}$$

$$S_j - S_i \geq -M + (p_i + M)x_{ij} \qquad \forall\, (i,j) \in V^2 \tag{3.5}$$

$$\sum_{(i,j)\in F^2} x_{ij} \geq 1 \qquad \forall\, F \in \Phi \tag{3.6}$$

$$x_{ij} \in \{0,1\},\ x_{ii} = 0 \qquad \forall\, (i,j) \in V^2 \tag{3.7}$$

$$S_i \geq 0 \qquad \forall i \in V \tag{3.8}$$

The three first sets of constraints (3.2)-(3.4), together with the integrality constraints (3.7), derive from the linear ordering problem and describe the allowed permutations of the activities: constraints (3.2) are the precedence constraints; constraints (3.3) and *transitivity constraints* (3.4) ensure that no cycle occur in the permutations. Inequalities (3.5) link the linear ordering variables with the starting time

variables as stated above. The resource constraints (3.6) state that, in any forbidden set $F \in \Phi$, at least one sequencing decision must be taken.

Actually, the number of required constraints (3.6) can be restricted by considering only the *minimal* or *undominated* forbidden sets: a forbidden set F is undominated if it is (i) minimum for the inclusion relation on $(\Phi, \subseteq)$ and (ii) an antichain for the precedence relation on V. However, even with this restriction, the number of constraints (3.6) may remain exponential in the number n of activities. As a consequence, it could be practically impossible to enumerate[1] the minimal forbidden sets and then to directly solve the linear programming relaxation of problem (3.1)-(3.8). The presence of big-M constraints (3.5), which make this relaxation weak, is another obvious drawback of this formulation.

Despite of these drawbacks, Demassey *et al.* [DEM 02a] have obtained tight lower bounds on the KSD benchmark instances from this linear relaxation. They use a constraint propagation module as preprocessing, to compute M values as small as possible in constraints (3.5) and to generate all the minimal forbidden sets of cardinality 2 or 3. The other resource constraints (3.6) of the mathematical programming formulation are then relaxed. In addition, many deductions made during the propagation mechanism are used in order to tighten the linear relaxation *a priori*. Indeed, the two RCPSP models, of constraint programming and of linear programming, are very close: they both are based on natural date variables and distance between tasks variables. Some variable domain reductions of the constraint processing correspond, for example, to the deduction of new precedence constraints. They directly result in variable fixing or bounding in the mathematical programming formulation. Four families of structural cutting planes for the linear relaxation are also proposed in [DEM 02a]. They are derived by translating redundant constraints deduced by constraint processing. Such an approach extends prior works on the single machine problem by Balas [BAL 85] and Dyer and Wolsey [DYE 90], and on the job-shop problem by Applegate and Cook [APP 91], who proposed effective cutting plane generation procedures for the sequence-based formulations of these problems.

3.1.2. *Resource flow*

Artigues [ART 03] proposed an alternative formulation of the resource constraints for the sequence-based model. The amount of resources in use during the process of an activity is freed at the end of this process, and becomes available to start an other activity. For each resource, this transfer can be formulated as a flow over the activity-on-node precedence network $G(V, E)$ from activity 0 (the source) to activity $n + 1$ (the sink).

1. A pseudo-polynomial algorithm is given in [STO 05] to enumerate all the minimal forbidden sets of a RCPSP (see also sections 1.6 and 7.2)

The mathematical programming formulation is identical to the previous model, except for resource constraints (3.6) that are replaced by the flow inequalities (3.9)-(3.11) of a flow model:

$$\sum_{j \in V} f_{ij}^k = b_{ik} \qquad \forall i \in V, \forall k \in \mathcal{R} \tag{3.9}$$

$$\sum_{j \in V} f_{ji}^k = b_{ik} \qquad \forall i \in V, \forall k \in \mathcal{R} \tag{3.10}$$

$$0 \leq f_{ij}^k \leq \min(b_{ik}, b_{jk}) x_{ij} \qquad \forall (i,j) \in V^2, \forall k \in \mathcal{R} \tag{3.11}$$

Variable f_{ij}^k gives the amount of resource k transferred from activity i at its completion, to activity j at its start. By constraints (3.11), this flow value is limited to the capacity $\min(b_{ik}, b_{jk})$ of arc (i,j) if the arc exists (i.e. if i precedes j or $x_{ij} = 1$), and to 0 otherwise. Constraints (3.9) and (3.10) are the usual inequalities of flow conservation (respectively the outgoing flow and the incoming flow on node i). Note that, in this model, the dummy consumptions b_{0k} and $b_{(n+1)k}$ are fixed to B_k for each resource k.

Compared to the previous model, this one contains mn^2 float variables more, but only a polynomial number of constraints and it does not require to compute the forbidden sets at first. Still, this formulation induces many symmetries[2] and its linear programming relaxation has not been exploited to date. However, this formulation was applied in several dedicated algorithms relative to robust scheduling. Artigues and Roubellat [ART 00] considered the problem of activity insertion in an existing schedule (see Algorithm 4). Schwindt [SCH 05] and Neumann *et al.* [NEU 03] considered a variant of the RCPSP with sequence-dependant setup times and used this model as a feasibility test. Lastly, Leus and Herroelen [LEU 04] used the resource flow as an indicator of the robustness of a schedule regarding the uncertainty of the processing times.

3.2. Time-indexed formulations

Many scheduling problem formulations are based on time discretization since they naturally describe the usage of the resources and the processing of the activities over time. In these models, we need to fix a planning horizon, denoted by T, which means that all jobs have to be completed by time T. Time from 0 to T is then divided into

2. Consider two activities 1 and 2 (1 precedes 2) with consumption 2 on a resource of capacity 3, then the two solutions $(2,0,2,0,2)$ and $(2,1,1,1,2)$ for vector $(f_{01}, f_{02}, f_{12}, f_{13}, f_{23})$ are equivalent, i.e. they give the same schedule $S_1 = 0$, $S_2 = p_1$, $S_3 = p_1 + p_2$

elementary periods of one unit-length. For each activity $j \in V$ and for each time period $t \in \mathcal{T} = \{0, \ldots, T\}$, a boolean variable y_{jt} indicates whether activity j is processed at time t. The precise relation between these time-indexed variables and the processing times of the activities differs sometimes from one model to another. Usualy, it is given by one of these 3 possible definitions:

- $y_{jt} = 1$ if and only if $S_j = t$ (activity j starts at time t)
- $y_{jt} = 1$ if and only if $S_j + p_j = t$ (activity j finishes at time t)
- $y_{jt} = 1$ if and only if $S_j \leq t$ (activity j starts before time t)

The resulting models are not significantly different and it is mostly trivial to translate a model into another following these definitions. In this section, the time-indexed formulations are given using the first definition above (also the most commonly used). Thus, we have relation:

$$S_j = \sum_{t=0}^{T} y_{jt}, \qquad \text{for each activity } j \in V.$$

Two different time-indexed modeling approaches have been proposed for the RCPSP and, again, the difference comes from the way the resource constraints are formulated. In a first formulation, one linear constraint for each time period is formulated to avoid resource conflicts (section 3.2.1). In the other formulation, resource conflicts are resolved at first, then stipulated as binary variables (section 3.2.2).

3.2.1. *Resource conflicts as linear constraints*

By means of the time-indexed variables, the set $\mathcal{A}_t$ of activities which are processed at at given time t is entirely characterized:

$$\mathcal{A}_t = \{j \in \mathcal{A} \mid \sum_{s=t}^{t+p_j-1} y_{js} = 1\}.$$

To avoid resource conflicts at time t, we have to ensure that, for each resource, the total consumption of the activities in $\mathcal{A}_t$ does not exceed the resource capacity. One of the first mathematical programming formulations of the RCPSP was given by Pritsker *et al.* ([PRI 69]):

$$\min \quad \sum_{t \in \mathcal{T}} t \cdot y_{(n+1)t} \tag{3.12}$$

$$\text{s.t.} \quad \sum_{t=0}^{T} y_{it} = 1 \qquad \forall\, i \in V \tag{3.13}$$

$$\sum_{t=0}^{T} t \cdot (y_{jt} - y_{it}) \geq p_i \qquad \forall\, (i,j) \in E \tag{3.14}$$

$$\sum_{i \in V} b_{ik} \sum_{\tau=t-p_i+1}^{t} y_{i\tau} \leq B_k \qquad \forall\, k \in \mathcal{R}, \forall\, t \in \mathcal{T} \tag{3.15}$$

$$y_{it} \in \{0,1\} \qquad \forall\, i \in V, \forall\, t \in \mathcal{T} \tag{3.16}$$

Constraints (3.13) impose that each activity is started exactly one time over the planning horizon T. Inequalities (3.14) and (3.15) are the precedence and the resource constraints, respectively.

The size of this binary integer program grows with the value of the time horizon T and then may be very large. However, a simple preprocessing, computing tight time-windows for the activities, makes it possible to greatly decrease the number of variables: $y_{jt} = 0$ if $t < ES_j$ or $t > LS_j$. The main drawback of this compact formulation is the low quality of the linear programming relaxation, as its solutions are far from being integral.

In [CHR 87], Christofides *et al.* restate of the precedence constraints (3.14) in a *disaggregated* manner as

$$\sum_{\tau=t}^{T} y_{i\tau} + \sum_{\tau=0}^{t+p_i-1} y_{j,\tau} \leq 1 \quad \forall\, (i,j) \in E, \forall t \in \mathcal{T}. \tag{3.17}$$

Replacing precedence constraints (3.13) by (3.17) makes the mathematical programming formulation theorically stronger since inequalities (3.17) and (3.13) together imply (3.14). Uetz [UET 01] shows on a specific instance of the RCPSP that, with disaggregated precedence constraints, the linear programming relaxation provides lower bounds that may be 75% higher. Yet, in practice, the additional computational time needed to solve this larger linear program does not seem to be counterbalanced by a significant improvement of the bound (see e.g. [CAV 01, MÖH 03, DEM 02a]).

Families of structural valid inequalities were proposed in [CHR 87, SAN 99, DEM 05] to tighten the time-indexed formulation. Among them, the *clique cuts* can be particularly strong and easily identified by analyzing the problem *a priori*. One way of generating clique cuts for the RCPSP is to compute disjunctive subproblems (or redundant single machines, see section 2.2): if $\mathcal{C}$ is a set of activities that can not overlap in time pairwise, then at most one activity in $\mathcal{C}$ is in process at any time t. This statement can be formulated as

$$\sum_{j\in\mathcal{C}} \sum_{\tau=t-p_j+1}^{t} y_{j\tau} \leq 1 \qquad \forall\, t \in \mathcal{T}. \tag{3.18}$$

This cutting plane procedure is applied in [DEM 05]. In a preprocessing phase, disjunctive subproblems are generated from greedy heuristics and used to run some constraint propagation techniques such as edge-finding. These algorithms detect new non-overlapping relationships which help, in turn, to generate new disjunctive subproblems of larger sizes, providing stronger clique inequalities for the linear program.

Fisher [FIS 73], Christofides et al [CHR 87] and Möhring *et al.* [MÖH 03] considered the lagrangian relaxation of the Pritsker's formulation, which is obtained by dualizing resource constraints (3.15). The lagrangian subproblem with the non-negative multipliers $(\lambda_{kt})_{k\in\mathcal{R},t\in\mathcal{T}}$ is

$$\min \sum_{t\in\mathcal{T}} t\cdot y_{(n+1)t} + \sum_{t\in\mathcal{T}}\sum_{i\in V}\Big(\sum_{k\in\mathcal{R}} b_{ik} \sum_{\tau=t}^{t+p_i-1} \lambda_{k\tau}\Big) y_{it} - \sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{R}} \lambda_{kt} B_k \tag{3.19}$$

s.t. (3.13), (3.17), (3.16)

Since the polyhedron defined by inequalities (3.13), (3.17) and $y \geq 0$ is integral, the optimum value of the lagrangian dual is not greater than – and *a fortiori* it equals – the linear relaxation bound of the disaggregated formulation. Nevertheless, solving the lagrangian dual with an incomplete iterative procedure, e.g. the subgradient algorithm, may yield a good approximation (from below) of this value, while, within the same time, the linear programming relaxation can not always be solved at all.

The lagrangian subproblems are instances of the project scheduling problem without resource constraints and with start-time dependent costs. The polynomial time complexity (the polyhedron is integral) of this problem was first exploited by Möhring *et al.* [MÖH 03]. In their procedure, each lagrangian subproblem is reduced to a minimum cut problem in a directed graph and solved in time $O(nmT^2\log(T))$. This procedure is a good trade-off between bound quality and computational time. Actually, in the experiments, the results (bounds and times) are similar to the linear programming relaxation of the aggregated solution. However, the procedure has a second purpose

since the solutions of the lagrangian subproblems are used to derive, with small additional efforts, feasible solutions for the RCPSP as well.

The paper by Möhring *et al.* [MÖH 03] is also a good reference for the computational study of the time-indexed formulation. The quality of the lower bounds and the computational times are compared for the linear programming relaxations (with aggregated or disaggregated precedence constraints, augmented or not with a family of clique cuts, solved with the primal simplex or the barrier algorithms, etc.) and for their own lagrangian relaxation approach. In Chapter 7, extensive computational experiments are carried out to evaluate the quality of the lower bounds obtained by the time-indexed formulations with and without constraint-propagation-based preprocessing.

3.2.2. *Feasible configurations*

Mingozzi *et al.* [MIN 98] built their model on the notion of *feasible set* of activities. A set $\phi \subseteq \mathcal{A}$ of activities is a feasible set if all the activities in ϕ can be processed simultaneously according to the precedence constraints (it is an antichain in the precedence graph $G(V, E)$) and according to the resource constraints (the total consumption does not exceed the capacity of any resource). Formally, $\phi \subseteq \mathcal{A}$ is a feasible set if and only if:

1) for all resource k, $\sum_{i \in \phi} b_{ik} \leq B_k$; and,

2) for any pair of activities (i, j) in ϕ, there is no path from i to j or from j to i in the precedence graph $G(V, E)$.

Let $l \in \Phi$ denote the indices of all feasible sets of $\mathcal{A}$, and among them, $l \in \Phi_i$ denote the indices of the feasible sets containing a given activity i.

In Mingozzi's formulation, resource conflicts are solved by ensuring that, on each unit time period, the activities in process form a feasible set. It is based on additional boolean time-indexed variables $(\zeta_{lt})_{l \in \Phi, t \in \mathcal{T}}$: variable ζ_{lt} takes value 1 if and only if all the activities of the feasible set ϕ_l are in process in time period $[t, t+1[$.

$$\min \quad \sum_{t=0}^{T} t y_{(n+1)t} \tag{3.12}$$

$$\text{s.t.} \quad \sum_{t=0}^{T} y_{it} = 1 \qquad \forall\, i \in V \tag{3.13}$$

$$\sum_{t=0}^{T} t(y_{jt} - y_{it}) \geq p_i \qquad \forall\, (i,j) \in E \tag{3.14}$$

$$\sum_{l \in \Phi_i} \sum_{t=0}^{T} \zeta_{lt} = p_i \qquad \forall\, i \in \mathcal{A} \tag{3.20}$$

$$\sum_{l \in \Phi} \zeta_{lt} \leq 1 \qquad \forall\, t \in \mathcal{T} \tag{3.21}$$

$$y_{it} \geq \sum_{l \in \Phi_i} \zeta_{lt} - \sum_{l \in \Phi_i} \zeta_{l(t-1)} \qquad \forall\, t \in \mathcal{T}, \forall\, i \in \mathcal{A} \tag{3.22}$$

$$\zeta_{lt} \in \{0,1\},\ \zeta_{l(-1)} = 0 \qquad \forall\, l \in \Phi, \forall\, t \in \mathcal{T} \tag{3.23}$$

$$y_{it} \in \{0,1\} \qquad \forall\, i \in V, \forall\, t \in \mathcal{T} \tag{3.16}$$

In this formulation, constraints (3.20), (3.21) and (3.22) replace the resource constraints (3.15) of the original time-indexed formulation. Constraints (3.20) ensure that any activity i appears in exactly p_i feasible sets processed on an unit time period. Constraints (3.21) impose that at most one feasible set is in process at any time. Constraints (3.22) link the feasible set variables ζ to the activity variables y, and ensure, together with constraints (3.13), the non-preemption of the activities. Note that the objective function has another formulation as:

$$\min \quad \sum_{l \in \Phi} \sum_{t \in \mathcal{T}} \zeta_{lt}. \tag{3.24}$$

This extended formulation was proposed together with a specific branching scheme that builds a solution by progressively sequencing such feasible blocks of activity parts (the branching scheme is described in section 5.1.3.2). The number of feasible sets is potentially exponential in the number of activities, hence this linear program may contain too many variables and constraints to directly provide a lower bound. Nevertheless, Mingozzi *et al.* have derived interesting lower bounds from this formulation by relaxing the non-preemption and the precedence constraints to disjunctions and time-windows. This study is the foundation for the current best lower

bounds for the KSD instances, by Brucker and Knust [BRU 00], and then by Baptiste and Demassey [BAP 04]. Also, Damay *et al.* [DAM 08] have recently applied such a relaxation to the preemptive RCPSP. Their approach is described in section 8.1.

3.2.2.1. *Combinatorial relaxations*

Mingozzi *et al.* obtain a first relaxation by aggregating the ζ variables corresponding to a same feasible set. Continuous variables $(z_l)_{l\in\Phi}$ are introduced and represent the time a given feasible set is executed in: $z_l = \sum_{t=0}^{T} \zeta_{lt}$. The linear program $(LB1)$ defined by (3.24) subject to (3.20) and (3.23) is rewritten as

$$\min \quad \sum_{l\in\Phi} z_l \tag{3.25}$$

$$s.t. \quad \sum_{l\in\Phi_i} z_l \geq p_i \qquad \forall i \in \mathcal{A} \tag{3.26}$$

$$z_l \geq 0 \qquad \forall l \in \Phi \tag{3.27}$$

By relaxing the integrality constraints (3.23) and the equality condition (3.20), Mingozzi *et al.* show that the optimum value of this program does not change when replacing Φ by $\Phi_{\max}$ the set of *undominated feasible subsets* (i.e. the feasible sets that are maximum for the inclusion order). They prove also that this bound, named $LB2$, is at least as good as the critical path length.

Naturally, the number of variables decreases but remains exponential in the worst case. The dual program of (3.25)-(3.27), which may be reduced to a weighted node packing problem, was considered and solved heuristically to yield a third bound $LB3$. This bound is used, in particular, in the branch-and-bound approach by Demeulemeester and Herroelen [DEM 97].

3.2.2.2. *Column generation and further improvements*

A different approach to tackle formulation (3.25)-(3.27) was proposed by Brucker and Knust [BRU 00] as an extension of their previous works published in [BAA 98, BRU 00]: the feasible sets are generated during the linear program solution within a column generation procedure. In order to reduce the search space of the feasible sets, the planning horizon $[0, T)$ is divided according to the time events $\{ES_i, LF_i \mid i \in \mathcal{A}\}$ – the release dates and deadlines of activities computed from the precedence relationship – into a set Δ of disjoint intervals. Then, the linear program is decomposed as well: at each iteration of the column generation procedure, and for each elementary interval $\delta \in \Delta$, a pricing subproblem generates new feasible sets involving only activities available on δ, i.e. activities $i \in \mathcal{A}$ such that $\delta \subseteq [ES_i, LF_i)$. Thus, in this decomposition, pricing results in solving small instances of the multidimensional knapsack problem, e.g. by branch-and-bound [BRU 00].

Linear program (3.25)-(3.27) is a relaxation of the preemptive RCPSP where precedence constraints are partially handled as disjunctions and via the time-windows. Actually, Brucker and Knust do not consider this optimization problem but its satisfaction form (S_T). Given an hypothetical upper bound T, the time-windows of the activities are computed, then the following linear program is solved by column generation:

$$\min \quad \sum_{\delta \in \Delta} u_\delta \tag{3.28}$$

$$s.t. \quad \sum_{\delta \in \Delta} \sum_{l \in \Phi_i^\delta} z_l^\delta \geq p_i \qquad \forall i \in \mathcal{A} \tag{3.29}$$

$$\sum_{l \in \Phi_i^\delta} z_l^\delta - u_\delta \leq L_\delta \qquad \forall \delta \in \Delta \tag{3.30}$$

$$z_l^\delta \geq 0 \qquad \forall \delta \in \Delta, \forall l \in \Phi^\delta \tag{3.31}$$

$$u_\delta \geq 0 \qquad \forall \delta \in \Delta \tag{3.32}$$

In this model, Φ^δ (respectively Φ_i^δ) denotes the set of feasible sets that can be processed in interval $\delta \in \Delta$ (respectively the ones which contain activity i) and L_δ denotes the length of interval δ. One variable z_l^δ is now defined for each interval $\delta \in \Delta$ and for each[3] feasible set $l \in \Phi^\delta$. The non-negative variables $(u_\delta)_{\delta \in \Delta}$ measure the overload of the intervals: the satisfaction problem (S_T) is feasible if and only if all these variables can be turned to 0, i.e. the optimum value of the linear program (3.28)-(3.32) is 0.

Thus, the whole procedure consists of successive attempts to solve the satisfaction problems (S_T) for different values T: the highest value T such that (S_{T-1}) is proved to be infeasible is a lower bound for the preemptive RCPSP and then for the original problem. This approach, called *destructive improvement approach*, was already used by Klein and Scholl [KLE 99] to get lower bounds for the RCPSP. Their experiments, comparing several combinatorial bounds for the RCPSP computed in a constructive and in a destructive way, show the benefit of this latter approach in general, for the higher quality of the bound and also for the reduced computation time. Of course, the key component of the destructive mechanism relies on the inner algorithm that tempts to refute an hypothetical upper bound T.

Note that the destructive approach can be seen as a basic and incomplete optimization procedure associated with a constraint propagation checker (the inner algorithm that checks the validity of T). Precisely, Brucker and Knust employ the mathematical

3. Observe that a feasible set may appear in several intervals, then in different variables.

program as a feasibility test within a constraint propagation procedure: on each trial value T, several well-known filtering (or domain reduction) rules are applied to a constraint programming model of (S_T), until a fixed point is reached. If (S_T) is proved to be inconsistent during this process (i.e. a variable domain is emptied), upper bound T is rejected. Otherwise, the time-windows of the activities, computed and tightened by this propagation, are provided to the column generation procedure.

3.2.2.3. *Cutting planes for the preemptive relaxation*

Baptiste and Demassey [BAP 04] improved upon this procedure by performing more constraint propagation – they built heavily loaded single redundant machines (see section 2.2) with the help of a binary integer program – and by introducing three families of structural cutting planes for the linear program (3.28)-(3.32). As in [DEM 05], some of these cuts are directly inspired by standard domain reduction rules, especially rules based on energetic reasoning. The energetic reasoning, as described in section 4.2.5, considers the minimum amount of time p_i^δ an activity i is required over an interval δ. This definition, transcribed for formulation (3.28)-(3.32), and applied to each sequence $\mathcal{D}$ of adjacent intervals $\delta \in \Delta$ (so $\mathcal{D}$ is assimilated to an interval), gives a first set of cuts:

$$\sum_{\delta \in \mathcal{D}} \sum_{l \in \Phi_i^\delta} z_l^\delta \geq p_i^{\mathcal{D}} \qquad \forall i \in \mathcal{A}, \forall \text{ interval } \mathcal{D} \subseteq \Delta. \tag{3.33}$$

A second family of cuts is defined in a very similar way to strengthen non-preemptivity in the formulation. Clearly, if the duration of a non-preemptive activity $i \in \mathcal{A}$ is not greater than the distance between two intervals $\delta, \delta' \in \Delta$, then i cannot be processed both in δ and δ'. The non-preemptivity cuts generalize this property to any set $\Psi \subseteq \Delta$ of intervals, such that the minimum distance L_Ψ between any two distinct intervals in Ψ is not lower than p_i:

$$\sum_{\delta \in \Psi} \sum_{l \in \Phi_i^\delta} z_l^\delta \leq \max\{L_\delta \mid \delta \in \Psi\} \qquad \forall i \in \mathcal{A}, \forall \Psi \subseteq \Delta \mid L_\Psi \geq p_i. \tag{3.34}$$

A last family of cuts aims to enforce the precedence constraints that are not totally handled within the formulation:

$$m_j - m_i \geq \frac{p_i + p_j}{2} \qquad \forall (i,j) \in E. \tag{3.35}$$

They introduce additional continuous variables $(m_i)_{i \in \mathcal{A}}$, figuring the mid-processing times of the activities, and which are defined (approximately) in the model, as follow:

$$\sum_{\delta \in \Delta} (t_\delta^- + \frac{1}{2}) \sum_{l \in \Phi_i^\delta} z_l^\delta \leq m_i p_i \leq \sum_{\delta \in \Delta} (t_\delta^+ - \frac{1}{2}) \sum_{l \in \Phi_i^\delta} z_l^\delta, \ \forall i \in \mathcal{A} \tag{3.36}$$

where $t_\delta^- = \min\{t \in \delta\}$ and $t_\delta^+ = \min\{t \in \delta\}$.

Observe that all these cutting planes do not change the pricing subproblem defined within the column generation procedure by Brucker and Knust [BRU 00]. Thus, Baptiste and Demassey have evaluated these cutting planes under nearly[4] the same destructive/constraint programming/column generation approach. Their experiments show the contribution of each of these cut families: improving lower bounds (also closing benchmark instances) or even, for some instances, reducing the computational time with few destructive iterations required.

3.3. Linear lower bounds and redundant resources

The mathematical programming formulation proposed by Carlier and Néron [CAR 03, CAR 07] is a kind of hybridation between a sequence-based model and Mingozzi's relaxation (3.25)-(3.27). An interpretation of this formulation is that we want to find a partitionning $[t_1 = 0, t_2), \ldots, [t_\Delta, t_{\Delta+1} = S_{n+1})$ of the scheduling time such that the length of each interval is minimal according to the number of time units during which the activities are executed in the interval. This minimum length is obtained by considering different CuSP subproblems, associated to each initial resources in $\mathcal{R}$ or to generated redundant resources (see section 2.4), and then by computing *linear lower bounds* on these CuSP. A linear lower bound for a CuSP is a lower bound of the makespan which is linearly dependant of the processing times of the activities. An example of a linear lower bound for the CuSP subproblem of the RCPSP associated to resource R_k is the basic $LLB_0(p) = \sum_{i \in \mathcal{A}} p_i b_{ik}/B_k$. It implies that the parts of activities, which are executed in a given interval $[t_\delta, t_{\delta+1})$, can be processed on the resource R_k only if

$$LLB_0(z^\delta) = \sum_{i \in \mathcal{A}} z_i^\delta \frac{b_{ik}}{B_k} \leq t_{\delta+1} - t_\delta \tag{3.37}$$

where z_i^δ denotes the duration of an activity $i \in \mathcal{A}$ within interval $[t_\delta, t_{\delta+1})$.

4. In [BAP 04] the pricing subproblems are solved with an integer linear programming solver. Also, the trial values T are tested in increasing order rather than by dichotomy.

The mathematical programming formulation is based on any set $\{LLB_1, \ldots, LLB_H\}$ of linear lower bounds that can be derived from the RCPSP:

$$\min \quad t_{\Delta+1} \tag{3.38}$$

$$s.t. \quad \sum_{\delta=1}^{\Delta} z_i^\delta = p_i \qquad \forall i \in \mathcal{A} \tag{3.39}$$

$$z_i^\delta \leq t_{\delta+1} - t_\delta \qquad \forall \delta \in \{1, \ldots \Delta\}, \forall i \in \mathcal{A} \tag{3.40}$$

$$LLB_h(z^\delta) \leq t_{\delta+1} - t_\delta \qquad \forall \delta \in \{1, \ldots, \Delta\}, \forall h \in \{1 \ldots, H\} \tag{3.41}$$

$$t_\delta \leq t_{\delta+1} \qquad \forall \delta \in \{1, \ldots, \Delta\} \tag{3.42}$$

$$z_i^\delta \geq 0 \qquad \forall \delta \in \{1, \ldots, \Delta\}, \forall i \in \mathcal{A} \tag{3.43}$$

$$t_\delta \geq 0 \qquad \forall \delta \in \{1, \ldots, \Delta + 1\} \tag{3.44}$$

In this model again, preemption is allowed and the precedence constraints are relaxed to time-windows in the CuSP subproblems involved in constraints (3.41). The quality of the lower bound obtained by solving this linear program depends entirely on the relevance of the CuSP subproblems (i.e. the redundant resources and the linear lower bounds) involved in constraints (3.41). It is of great interest to apply such approach in a branch-and-bound procedure for the RCPSP, since the main effort, to generate the CuSP subproblems, can be made once at the root of the tree search. Then the linear program can be run, in polynomial time, at each node of the tree search.

Thus, this approach is a very promising attempt to close the gap between the two categories of lower bounds which exist for the RCPSP: the fast computable combinatorial relaxations (Chapter 2) and the tight mathematical programming-based lower bounds (Chapter 3). This is one of the most challenging issues for future research on solving the RCPSP at optimality.

Chapter 4

Constraint Programming Formulations and Propagation Algorithms

This chapter presents the classical representation of RCPSP in constraint-based scheduling as well as the propagation algorithms for temporal constraints and resource capacity constraints (time-tabling, disjunctive reasoning, edge-finding, energetic reasoning, precedence energy and balance) used to solve the RCPSP through constraint-based scheduling.

4.1. Constraint formulations

4.1.1. *Constraint programming*

Constraint programming (CP) is based on a declarative description of a problem as (1) a set of decision variables with domains, i.e., sets of possible values, for instance the start time of an activity with its possible time-windows; and (2) a set of constraints restricting the combinations of values the decision variables can take, for instance a precedence constraint between the end time of activity A and the start time of activity B.

Constraints can be defined either explicitly as a set of compatible tuples of variable assignments or implicitly, for instance through an algebraic relation (e.g. $x \neq y$). A solution to a CP problem is a complete assignment of the variables satisfying all the constraints.

Chapter written by Philippe LABORIE, Wim NUIJTEN.

The search for a solution usually explores a search tree whose branches correspond to decisions (for instance instantiating a variable x to a given value v). During the search, constraints are used actively to remove inconsistent values from the current domain of variables. This process is known as *constraint propagation* [BES 06]. One way of propagating constraints consists of enforcing arc-consistency. A constraint is said to be arc-consistent if the domains of its variables have been reduced so that they only contain values that are supported by at least one solution to the constraint. For instance if the domain of variables x and y in constraint $C = (x \leq y)$ are so that $x \in [5, 10]$ and $y \in [0, 7]$, enforcing arc-consistency on C will reduce these domains to $x \in [5, 7]$ and $y \in [5, 7]$. At each node explored by the search, constraints are propagated until the fix point of the propagation is reached.

A particularly important class of constraints are the *global constraints*. Such constraints represent a set of basic constraints (for instance $\texttt{alldifferent}(x_1, ..., x_n)$ standing for $\forall i \in [1, n], j \in [i + 1, n], x_i \neq x_j$). Algorithms for propagating global constraints are able to propagate from a global point of view in an efficient way. The typical scheduling example of such a global constraint is the constraint that propagates on the combination of all activities requiring capacity from a non-renewable resource.

4.1.2. *Constraint-based scheduling*

Constraint-based scheduling is the discipline that studies how to solve scheduling problems by using CP. Over the years, it has grown into one of the most successful application areas of CP. Constraint-based scheduling extends CP by providing constructs such as activities and resources which have a scheduling semantics. An activity usually corresponds to at least 3 decision variables: start time, end time and processing time or duration. The domains of these variables represent the possible time-windows and duration of the activity. Ususal constraints in constraint-based scheduling are temporal constraints between activities and resource capacity constraints. Different types of resources can be modeled [BAP 06]: unary resources, cumulative resources, producible/consumable resources, state resources, etc. Resource demands represent a certain demand of resource (this quantity can be a decision variable) for a given activity. Although most of all the extensions described in Part II of this book can be modeled and solved using the paradigm of constraint-based scheduling, we focus in this chapter on the basic formulation of RCPSP and the underlying constraint propagation algorithms. More detailed surveys of constraint propagation algorithms for constraint-based scheduling are available in [DOR 99, BAP 01].

The basic RCPSP can be formulated, as shown in Table 4.1.2, in a constraint-based scheduling approach using OPL, the Optimization Programming Language [VAN 99]. The precedence graph structure is defined in line `8` as a set of precedence arcs `i,j`. The set of activities is defined in line `11`, activity `A[i]` being constructed with duration `p[i]`. An additional activity of duration `0` representing the makespan is defined in

line 12. The set of cumulative resources is created in line 13, resource R[k] being of capacity B[k]. Discrete resources post a global capacity constraint on the set of activities that require the resource. The objective function (minimize end time of makespan activity) is specified in line 14 followed by the constraints of the problem. Each activity A[i] is constrained to finish before the makespan activity (line 17) and to require b[i][k] units of resource R[k] in line 19. Finally, the precedence constraints of the precedence graph G are posted in line 22.

```
1  // Data
2  struct Precedence { int i; int j; };
3  int q            = ...; // Number of resources
4  int n            = ...; // Number of activities
5  int p[1..n]      = ...; // Processing times
6  int B[1..q]      = ...; // Resource capacities
7  int b[1..n,1..q] = ...; // Resource requirements
8  {Precedence} G   = ...; // Precedence graph
9
10 // Model
11 Activity A[i in 1..n](p[i]);
12 Activity makespan(0);
13 DiscreteResource R[k in 1..q](B[k]);
14 minimize makespan.end
15 subject to {
16   forall(i in 1..n) {
17     A[i] precedes makespan;
18     forall(k in 1..q : 0 < b[i][k])
19       A[i] requires(b[i][k]) R[k];
20   };
21   forall(<i,j> in G)
22     A[i] precedes A[j];
23 };
```

Table 4.1. *An OPL model of the basic RCPSP*

The next section describes the principles of the classical algorithms for propagating temporal constraints (section 4.2.1) and resource constraints (sections 4.2.2-4.2.8).

4.2. Constraint propagation algorithms

4.2.1. *Temporal constraints*

Temporal relations between activities can be expressed by linear constraints between start and end variables of activities. For instance, a standard precedence constraint between two activities A_i and A_j stating that A_j is to be started after A_i has ended can be modeled by the linear constraint $C_i \leq S_j$. In the general case of RCPSP with time lags, with both x and y a start or end variable and δ an integer, temporal relations can be expressed by constraints of the type $x - y \leq \delta$.

When the temporal constraint network is sparse, as it is usually the case in RCPSP, such constraints can be easily propagated using a standard arc-consistency algorithm [LHO 93]. In addition, a variant of Ford's algorithm (see for instance [GON 84]) proposed by Cesta and Oddi [CES 96] can be used to detect any inconsistency between such constraints in time polynomial to the number of constraints and independent of the domain sizes.

When the temporal network is dense or when it is useful to compute and maintain the minimal and maximal delay between any pair of time points in the schedule, path consistency can be enforced on the network [DEC 91] for example by applying Floyd-Warshall's All-Pairs-Longest-Path algorithm [FLO 62].

4.2.2. *Timetabling*

Timetabling relies on the computation of the minimal resource usage by the current activities in the schedule at every date t [Le 94b]. This aggregated demand profile is maintained during the search. It allows restricting the domains of the start and end times of activities by removing the dates that would lead to an over-consumption of the resource.

Suppose that activity A_i requires $b_{ik} \in [b_{ik}^-, b_{ik}^+]$ units of resource R_k and is such that $LS_i < EC_i$. Then we know that A_i will surely execute over the time interval $[LS_i, EC_i)$ requiring at least b_{ik}^- units of R_k; see activity A_1 in Figure 4.1 for an illustration. The minimal contribution of activity A_i at time t is thus given by the function:

$$U_k(A_i, t) \quad = \quad \begin{cases} b_{ik}^- & \text{if } LS_i \leq t < EC_i \\ 0 & \text{otherwise} \end{cases}$$

For each resource R_k, a curve $U_k(t)$ is maintained that aggregates all these demands: $U_k(t) = \sum_{i=1}^{n} U_k(A_i, t)$.

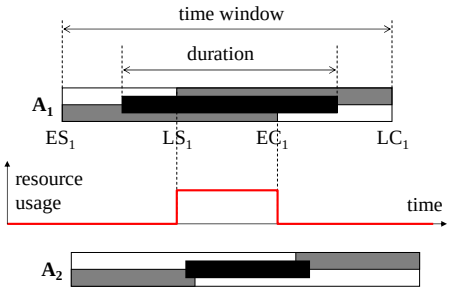


Figure 4.1. *Timetabling*

It is clear that if there exists a date t such that $U_k(t)$ is strictly greater than the maximal capacity of the resource B_k, the current schedule cannot lead to a solution and the search must backtrack .

Furthermore, $U_k(t) - U_k(A_i, t)$ represents the requirement by all activities except A_i on R_k at date t. If there exists a date t such that $U_k(t) - U_k(A_i, t) + b_{ik}^- > B_k$, then activity A_i cannot be executing at time t. In particular, if the above relation is true for some t with $ES_i \leq t < EC_i$ (resp. $LS_i \leq t < LC_i$), it allows to increase (resp. decrease) the earliest start time (resp. latest completion time) of activity A_i until the first date t' such that $U_k(t') - U_k(A_i, t') + b_{ik}^- \leq B_k$. Similar reasoning can be applied to find new upper bounds on the quantity of resource required by activities. The main advantage of the timetabling technique is its low practical algorithmic complexity, which is linear or even sub-linear in the number of activities. For this reason, it is the main technique used today for scheduling renewable resources on large problems.

4.2.3. *Disjunctive reasoning*

Let A_i and A_j be two activities on a resource R_k such that $b_{ik}^- + b_{jk}^- > B_k$. As such, they cannot overlap in time and thus either A_i precedes A_j or A_j precedes A_i, *i.e.*, the disjunctive constraint holds between these activities. In general the disjunctive constraint achieves arc-consistency on the formula:

$$[b_{ik} + b_{jk} \leq B_k] \vee [C_i \leq S_j] \vee [C_j \leq S_i]$$

The disjunctive constraint can be propagated on all activities with a worst case complexity in $O(n \log n)$ [VIL 04].

4.2.4. *Edge-finding*

Let $\phi \subset \mathcal{A}$ be a subset of activities that require a given resource R_k of capacity B_k and $A_i \in \mathcal{A} \setminus \phi$. Let us denote:

– $ES_\phi = \min_{A_j \in \phi} ES_j$, the earliest start time of all activities in ϕ;

– $EC_\phi = \min_{A_j \in \phi} EC_j$, the earliest completion time of all activities in ϕ;

– $LC_\phi = \max_{A_j \in \phi} LC_j$, the latest completion time of all activities in ϕ;

– $W_\phi = \sum_{A_j \in \phi} p_j \cdot b_{jk}$, the global energy required by ϕ.

The basic idea of edge-finding and activity interval techniques is to ensure that for any subset of activities ϕ, resource R_k provides enough energy over the time interval $[ES_\phi, LC_\phi)$ to allow the execution of all the activities of ϕ, that is: $W_\phi \leq B_k \cdot (LC_\phi - ES_\phi)$. Constraint propagation is usually performed by applying the three following deduction rules:

1) If $B_k \cdot (LC_\phi - ES_{\phi \cup \{i\}}) < W_{\phi \cup \{A_i\}}$, then A_i must finish after all activities in ϕ, in particular $EC_i := max(EC_i, EC_\phi)$.

2) If $ES_\phi < ES_i < EC_\phi$ and $B_k \cdot (LC_\phi - ES_\phi) < W_\phi + b_{ik} \cdot (min(LC_\phi, EC_i) - ES_i)$, then at least one activity A_j in ϕ must precede A_i, otherwise A_i would start between ES_ϕ and EC_ϕ and there would not be enough energy to execute $\phi \cup \{A_i\}$ between ES_ϕ and LC_ϕ, thus $ES_i := max(ES_i, EC_\phi)$.

3) If $ES_i < ES_\phi < EC_i$ and $B_k \cdot (LC_\phi - ES_\phi) < W_\phi + b_{ik} \cdot (EC_i - ES_\phi)$, then A_i must finish after all activities in ϕ, in particular $EC_i := max(EC_i, EC_\phi)$.

These rules make it possible to update the earliest start or completion times of activities and symmetrical rules allow to update the latest start and completion times. Edge-finding ([NUI 94]) and activity intervals propagation ([CAS 96]) are very similar techniques that mainly differ in the way the propagation rules are triggered: edge-finding algorithms are global algorithms that perform all updates on a given resource whereas activity interval approaches are performed incrementally as soon as the time bound of a activity changes.

Note that specific edge-finding algorithms have been designed for the special case of disjunctive resources ($B_k = 1$) [BAP 01]. These algorithms can be implemented with a time complexity in $O(n \log n)$ [VIL 04].

4.2.5. *Energy reasoning*

Energy-based constraint propagation algorithms compare the amount of energy provided by a resource over some interval $[t_1, t_2)$ to the amount of energy required by activities that have to be processed over this interval. [ERS 91] proposes the following definition of the required energy consumption that takes into account the fact that activities cannot be interrupted. Given a resource R_k, an activity A_i and a time interval $[t_1, t_2), W_k(A_i, t_1, t_2)$, the *left-shift / right-shift* required energy consumption of resource R_k by activity A_i over $[t_1, t_2)$ is b_{ik} times the minimum of the three following durations.

– the length of the interval: $t_2 - t_1$;

– the number of time units during which A_i executes after time t_1 if A_i is left-shifted, *i.e.*, scheduled as soon as possible: $p_i^L(t_1) = \max(0, p_i - \max(0, t_1 - ES_i))$;

– the number of time units during which A_i executes before time t_2 if A_i is right-shifted, *i.e.*, scheduled as late as possible: $p_i^R(t_2) = \max(0, p_i - \max(0, LC_i - t_2))$.

This leads to $W_k(A_i, t_1, t_2) = b_{ik} \cdot \min(t_2 - t_1, p_i^L(t_1), p_i^R(t_2))$.

The left-shift / right-shift overall required energy consumption $W_k(t_1, t_2)$ over an interval $[t_1, t_2)$ is then defined as the sum over all activities A_i of $W_k(A_i, t_1, t_2)$.

It is obvious that if there is a feasible schedule, then $B_k \cdot (t_2 - t_1) - W_k(t_1, t_2) \geq 0$ for all t_1 and t_2 such that $t_2 \geq t_1$.

It is shown in [BAP 01] how the values of W_k can be used to adjust activity time bounds. Given an activity A_i and a time interval $[t_1, t_2)$ with $t_2 < LC_i$, it is examined whether A_i can end before t_2. If there is a time interval $[t_1, t_2)$ such that

$$W_k(t_1, t_2) - W_k(A_i, t_1, t_2) + b_{ik} \cdot p_i^L(t_1) > B_k \cdot (t_2 - t_1),$$

then a valid lower bound of the end time of A_i is

$$t_2 + \frac{1}{b_{ik}}(W_k(t_1, t_2) - W_k(A_i, t_1, t_2) + b_{ik} \cdot p_i^L(t_1) - B_k \cdot (t_2 - t_1)).$$

Similarly, when

$$W_k(t_1, t_2) - W_k(A_i, t_1, t_2) + b_{ik} \cdot \min(t_2 - t_1, p_i^L(t_1)) > B_k \cdot (t_2 - t_1),$$

A_i cannot start before t_1 and a valid lower bound of the start time of A_i is

$$t_2 - \frac{1}{b_{ik}}(C(t_2 - t_1) - W_k(t_1, t_2) + W_k(A_i, t_1, t_2)).$$

[BAP 01] presents a $O(n^3)$ algorithm to compute these time bound adjustments for all n activities. It is first shown there are $O(n^2)$ intervals $[t_1, t_2)$ of interest. Given an interval and an activity, the adjustment procedure runs in $O(1)$. As such, the obvious overall complexity of the algorithm is thus $O(n^3)$. An interesting open question is whether there is a quadratic algorithm to compute all the adjustments on the $O(n^2)$ intervals under consideration. Another open question at this point is whether the characterization of the $O(n^2)$ time intervals in [BAP 01] can be sharpened in order to eliminate some intervals and reduce the practical complexity of the corresponding algorithm. Finally, it seems reasonable to think that the time bound adjustments could be sharpened. Even though the energy tests can be limited (without any loss) to a given set of intervals, it could be that the corresponding adjustment rules cannot.

4.2.6. *Precedence graph*

The above constraint propagation algorithms reason on the absolute position of activities in time (through their time bounds ES_i, EC_i, LS_i, LC_i). They provide an efficient propagation when these time bounds are tight but the propagation becomes weaker when time-window of activities is large.

Some additional propagation algorithms have been designed that reason on the relative position of activities on resources (precedence relations in a precedence graph) rather than their absolute position only. As a consequence, these algorithms allow a much stronger propagation when the time-windows of activities are large and when the current schedule contains many precedence relations as it is usually the case in RCPSP. These algorithms require that a temporal network representing the relations between the time-points of all activities (start and end) using the point algebra of ([VIL 86]) is maintained during the search. We denote $\{\emptyset, \prec, \preceq, =, \succ, \succeq, \neq, ?\}$ the set of qualitative relations between the time points of the schedule S_i, C_i. The temporal network is in charge of maintaining the transitive closure of those relations. Let r be one of the above relations, we denote $\neg(x\ r\ y)$ if and only if the relation $x\ r\ y$ cannot be deduced from the network.

The initial set of relations consists of the precedences $S_i \prec C_i$ for each activity A_i and $C_i \preceq S_j$ for each precedence constraint of the problem. During the search additional precedence relation can be added as decisions or as the result of constraint propagation.

Figure 4.2 illustrates an example of such a precedence graph. An arrow between two time-points x and y means $x \prec y$. Once the transitive closure of the graph has been computed, the following relations hold: $S_3 \prec C_2$, $\neg(S_1 \prec C_4)$.

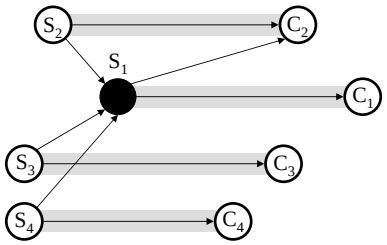


Figure 4.2. *Precedence graph*

4.2.7. *Energy precedence*

The *energy precedence* propagation algorithm ([LAB 03a]) for an activity A_i on a resource R_k ensures that for each subset Ω of predecessor activities of activity A_i the resource provides enough energy to execute all activities in Ω between ES_Ω and S_i. More formally, if $ES_\Omega = \min_{A_j \in \Omega} ES_j$ and $W_{\Omega,k} = \sum_{A_j \in \Omega} p_j \cdot b_{ik}$, it performs the following deduction rule:

$$\forall \Omega \subset \{A_j \in \mathcal{A}, C_j \preceq S_i\}, ES_i := max(ES_i, ES_\Omega + \lceil W_{\Omega,k}/B_k \rceil)$$

The propagation of the energy precedence constraint can be performed for all the activities A_i on a resource and for all the subsets Ω with a total worst-case time complexity of $O(n(p + log(n))$, where n is the number of activities on the resource and p the maximal number of predecessors of a given activity in the temporal network ($p < n$).

4.2.8. *Balance constraint*

On a renewable resource, the *balance constraint* ([LAB 03a]) can be defined as follows. The basic idea of the algorithm is to compute, for each activity A_i on a resource R_k, a lower bound on the resource usage at the start time of A_i (a symmetrical reasoning can be applied to perform some propagation based on a lower bound on the resource usage at the completion time of A_i). Using the temporal network a lower bound on the resource utilization at date $S_i + \epsilon$ just after the start time of A_i can be computed assuming that all the resource requirements that do not necessarily overlap S_i will not overlap it:

$$L_k(i) = \sum_{j/(S_j \preceq S_i) \wedge (C_j \succ S_i)} b_{jk}$$

Given this bound, the balance constraint is able to discover three types of information:

Dead ends. Whenever $L_k(i) > B_k$, the resource will surely be over-consumed just after time-point S_i so the search has reached a dead end.

New bounds on time variables. If $L_K(i) \leq B_k$, $\Delta(i) = B_k - L_K(i)$ represents a slack of capacity that must not be exceeded by all the resource requirements that, currently, do not necessarily overlap S_i but could overlap it. Let $P(i) = \{A_j / (S_j \preceq S_i) \wedge \neg(C_j \succ S_i)\}$. We suppose the activities $(A_{j_1}, \cdots, A_{j_u}, \cdots, A_{j_p})$ in $P(i)$ are ordered by decreasing earliest completion time EC_j. Let v be the index in $[1, p]$ such that:

$$\sum_{u=1}^{v-1} b_{j_u k} \leq \Delta(x) < \sum_{u=1}^{v} b_{j_u k}$$

If event S_i is executed at a date $S_i < EC_{j_v}$, not enough activities of $P(i)$ will be able to be completed strictly before S_i in order to ensure the resource is not over-consumed just after S_i as in this case, the consumed quantity will be at least $L_K(i) + \sum_{u=1}^{v} b_{j_u k} > B_k$. Thus, EC_{j_v} is a valid lower bound of S_i.

New precedence relations. Suppose there exists activity A_y in $P(i)$ such that:

$$\sum_{A_z \in P(i), C_z \succeq C_y} b_{zk} > \Delta(x)$$

Then, if we had $S_i \prec C_y$, we would see that again there is no way to avoid a resource over-consumption as it would consume at least:

$$L_k(i) + \sum_{A_z \in P(i), C_z \succeq C_y} b_{zk} > B_k$$

Thus, the necessary precedence relation: $C_y \preceq S_i$ can be deduced and added to the current temporal network.

On the precedence graph of Figure 4.2, if all activities require 1 unit of the same resource R_k of capacity 3, the balance constraint applied at time-point S_1 would compute $L_k(1) = b_{1k} + b_{2k} = 2$ and deduce that $\min(EC_3, EC_4) \leq S_1$.

The balance algorithm can be executed for all activities A_i with a global worst-case complexity in $O(n^2)$ if the propagation that discovers new precedence relations is not turned on and in $O(n^3)$ for a full propagation. In practice, there are many ways to shortcut this worst case and in particular, it was noticed that the algorithmic cost of the extra-propagation that discovers new precedence relations was in general negligible.

4.3. Conclusion

Constraint-based scheduling provides a wide spectrum of propagation algorithms for renewable resources ranging from sub-linear to quadratic time complexities. The

selection of a combination of propagation algorithms clearly depends on the size of the problem being solved.

For small problems until a few tens of renewable resources and a few hundred activities on each resource, a strong propagation scheme may be used. This is for instance the case of the MCS method described in section 7.6 and experimented on small RCPSPs. It uses a combination of timetabling, disjunctive, edge-finding, energy precedence and balance constraints.

Larger problems are generally solved using meta-heuristics. These methods, for instance the LNS described in section 7.6, are usually less sensitive to constraint propagation than tree search methods and light propagation schemes can be used, typically, only using timetabling.

Chapter 5

Branching schemes for Branch-and-Bound

In this chapter we present exact methods for solving the RCPSP, restricting to branch-and-bound methods. Linear programming based methods are described in Chapter 3. Constraint Programming methods (CP) are closed to the ones that are presented in this section. For instance, many CP filtering methods used to adjust time-bounds of activities are used into branch-and-bound methods. Moreover in this chapter, only branching schemes are presented: lower bounds, upper bounds and filtering methods are respectively described in Chapters 2, 6 and 4.

Notice that exact methods, of course are used to obtain an exact solution to the RCPSP but they are also helpful to obtain efficient heuristic solutions. A search tree can be truncated in order to limit the search to promising solutions, and recently they also were used as a part of heuristic based on large neighborhood search.

Branch-and-bound methods build a search tree in order to explore implicitly the search space. At each node of the search tree, two situations may occur: (1) a leaf node is reached and a feasible solution can be deduced, (2) after computing lower bounds, upper bounds and possibly time-bound adjustments, the search space corresponding to the current node is partitioned into subsets such that the union of these subsets corresponds to the set of solutions of the current node. This operation of partitioning an area of the search space into sub-areas is called *branching* (see Figure 5.1).

Building a search tree that efficiently explores all relevant solutions is a challenging problem. From the earliest works on the RCPSP [JOH 67], many authors have proposed several branching schemes. Most of the time, linked to branching schemes,

Chapter written by Emmanuel NÉRON.

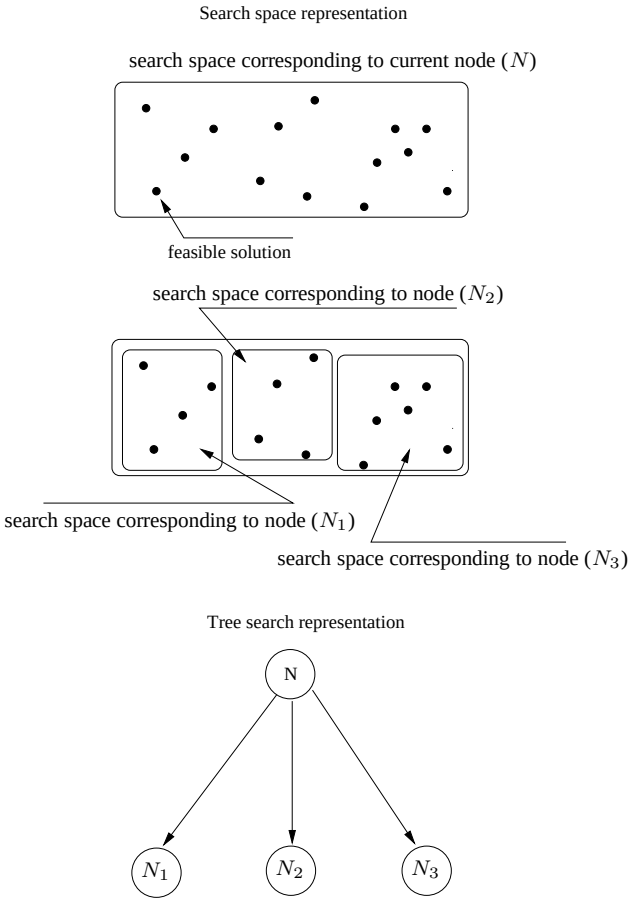


Figure 5.1. *Search space and tree search representation*

some dominance rules can be applied to detect that a given node is dominated and then can be pruned. The efficiency of a branching scheme is usually linked to both lower bounds (see Chapter 2 and 3) and dominance rules.

Figure 5.2 presents an RCPSP instance made up of 8 activities and 1 resource, having a capacity equal to 3. This instance is used in this chapter to illustrate how the search trees corresponding to the branching schemes that are presented are built.

Before describing main branching schemes that are used in other works for solving the RCPSP, note that the list of the branching scheme presented is not exhaustive. Here

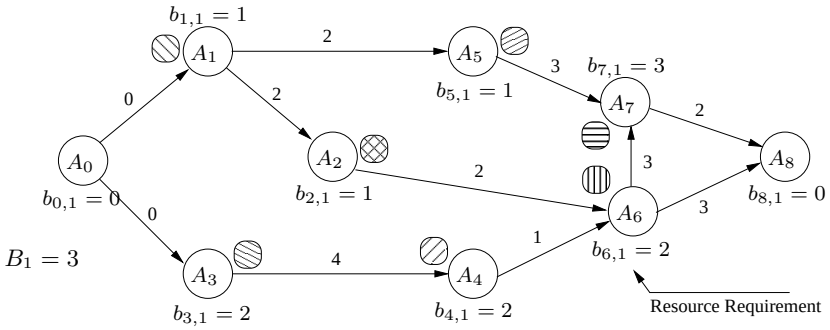


Figure 5.2. *RCPSP instance*

we present some branching schemes that from our point of view seem to be the most relevant in understanding how the solution space can be split and explored. However, notice that, for instance, several methods based on CP, that basically build search tree to explore the solution space and then can be seen as branch-and-bound have also been proposed in other works, e.g., [DOR 00, CAS 97, LAB 05, LIE 08].

5.1. Chronological branching scheme

The most natural way to build solution is to associate each node of the search tree with a partial solution, i.e., a partial schedule, and the branching step consists of adding at least one activity to this partial solution. Thus, a leaf of such a tree corresponds to a feasible solution. The main advantage of building and exploring search tree using this method is that partial schedule from time 0 to the current time-point is fixed for this subtree. Thu, testing that a partial solution corresponds to semi-active and/or active schedule [SPR 95] can easily be done at each node of the search tree in order to reduce the search space.

We present 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 search tree, whereas in the second feasible subsets of activities are added to the partial solution.

5.1.1. *Adding one activity to a partial solution*

5.1.1.1. *Considering all relevant activities*

Let N be a given node of the search tree that corresponds to a partial schedule, the most natural way to extend this partial solution is to consider the set of eligible

activities, EL, that are activities whose predecessors are already scheduled. Then one node is created for each activity in EL and this activity is added to the partial solution as soon as possible, at a time point greater than the starting time of the activity scheduled at the previous level, and respecting both the resource constraints and the precedence constraints (see for instance [PAT 90, SPR 00]).

Figure 5.3 presents how such a tree can be built on the instance presented above (see Figure 5.2). The root node corresponds to an empty schedule. Then two nodes are created due to the fact that A_1 and A_3 have no predecessor (the dummy start node of the project is ignored). If we focus on the left-node corresponding to the partial schedule made up of A_1, then the activities whose predecessors have been scheduled are $\{A_2, A_5, A_3\}$: three nodes are created each of them corresponds to adding one of these activities to the partial solution.

The corresponding partial schedules are semi-active (see Chapter 1) due to the fact that each time an activity is added, it is scheduled as soon as possible respecting both resource and precedence constraints, but after the starting time of the previously scheduled activity. Dominance rules have been proposed to reduce the search tree to active partial schedules (see section 5.1.2), and to avoid node redundancy: if A_i and A_j start at the same time in a partial schedule that has been built by scheduling A_j immediately after A_i then scheduling A_i immediately after A_j leads to the same partial schedule. For instance, in Figure 5.3, a node that is built from node (1) by adding activity A_1 to the partial schedule leads to the same partial solution than the one corresponding to node (2), and then it is not relevant.

5.1.1.2. *Delaying one activity*

Baptiste et al. [BAP 99], have proposed another method to build a partial schedule, adding one activity at a time. Basically, a given node N corresponds to a partial schedule. The eligible set of activities (EL) is defined as previously: it is the set of activities whose all predecessors have been already scheduled. Then one activity $A_i \in EL$ is chosen (the one with the minimum earliest starting time), and two nodes are created. In the first one the chosen activity is scheduled as soon as possible, in the second one, we consider that A_i cannot be the first activity of EL to be scheduled, and then we enforce at least one activity in $EL \setminus \{A_i\}$ to start before or simultaneously with the start of A_i. One immediate deduction is that ES_i can be updated: $ES_i \leftarrow max(ES_i, min_{A_j \in EL \setminus \{A_i\}} ES_j)$.

Partial solutions built using this branching scheme are not necessarily semi-active. Thus each time an activity is scheduled, checking whether the partial schedule is still semi-active becomes crucial.

This branching scheme is used to build the search tree presented in figure 5.4.

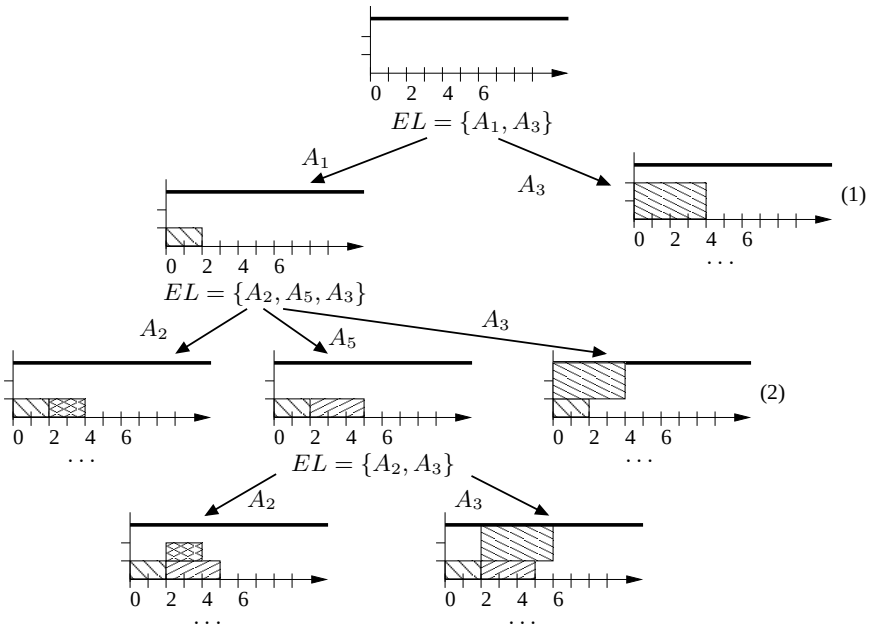


Figure 5.3. *Tree search based on chronological branching scheme*

5.1.2. *Dominance rule: left-shift and global left-shift*

According to the fact that semi-active and active schedules are dominant (see Chapter 1), dominance rules have been proposed in order to restrict the search space to the exploration of solutions that are active or at least semi-active. Testing that a partial solution is semi-active can be easily done each time an activity is added to the partial solution. Let S_i be the starting time of activity A_i. If at time $S_i - 1$ the activity can be scheduled without violating neither the precedence ($S_i - 1 \geq C_j, \forall A_j \in \Gamma_i^{-1}$), nor the resource constraints, the node that consists of adding activity A_i at time S_i can be pruned. Testing that a schedule is active may be time consuming ($O(n \cdot |\mathcal{R}|)$).

Sprecher [SPR 00] has proposed the following dominance rule, that extends the notion of active schedule. Once an activity $A_{i_{max}}$ is scheduled, we look for Δ, the largest amount of time between the end of its latest scheduled predecessor and $S_{i_{max}}$, during which the amount of resources required by $A_{i_{max}}$ are available for its processing. We denote by $A^*_{i_{max}}$, the activity that ends just before $C_{i_{max}}$. If Δ is greater or equal to $p_{i_{max}}$, $A_{i_{max}}$ can be inserted and the partial solution is not active. Otherwise, to insert $A_{i_{max}}$, a set of activities needs to be right shifted of at most $p_{i_{max}} - \Delta$ time-units. Thus, if the end of the partial schedule, given by $C_{i^*_{max}}$, once $A_{i_{max}}$ is

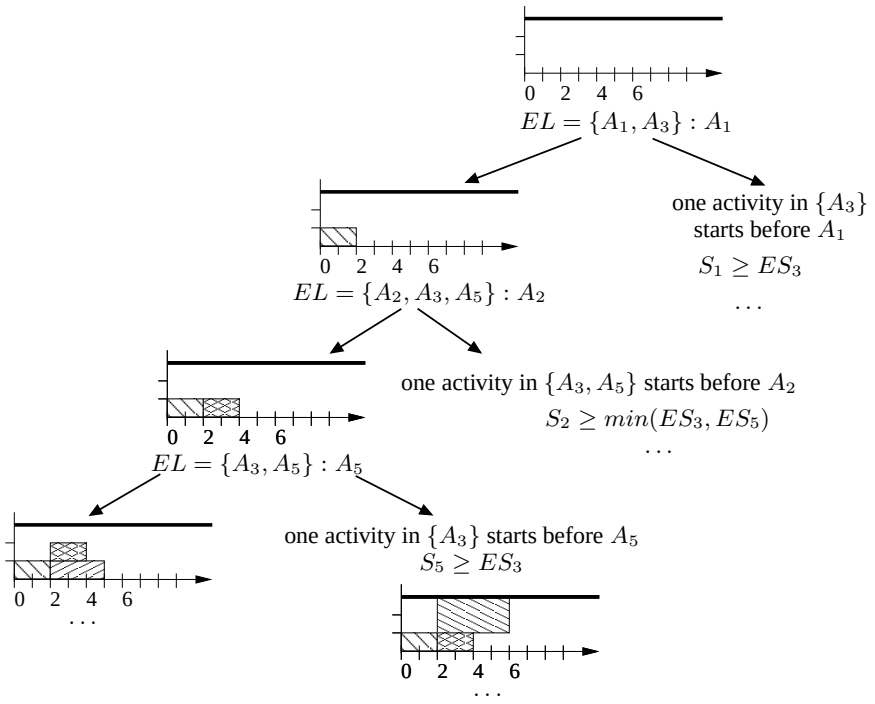


Figure 5.4. *Search tree based on chronological/delaying branching scheme*

inserted and the activities are right shifted, is strictly lower than the end of the current partial schedule, the current node is dominated and can be pruned. This dominance rule called *global left-shift* is illustrated in Figure 5.5

5.1.3. *Adding a subset of activities to a partial solution*

In this section, we present two branching schemes that have been proposed in the literature, that consist in adding more than one activity to a partial solution. Notice that branch-and-bound methods based on these branching schemes are among the most efficient ones specifically if used for solving classical instances of the literature.

5.1.3.1. *Delaying alternatives*

This branching scheme has been initially proposed by Christophides et al. [CHR 87]. Demeulemeester and Herroelen [DEM 97] proposed corrections and improvements. Let us consider a node given by a partial schedule, and a relevant

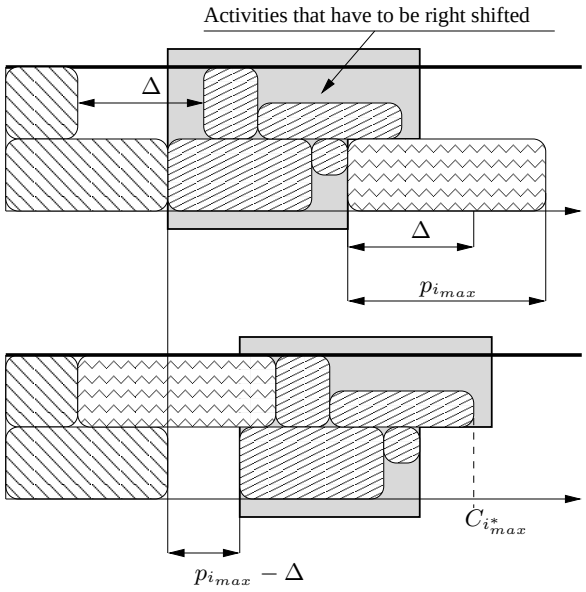


Figure 5.5. *illustration of global left-shift*

time-point t that corresponds to the earliest end of an activity scheduled at a previous level. Then the set $IP(t)$ of *in progress* activities at the considered time-point and $EL(t)$ of *eligible activities* at time t, are built. 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 of them. A delaying alternative, MDA, is a subset of $IP(t) \cup EL(t)$ such that if the activities of the MDA are delayed then the resource conflict disappears. A MDA is minimal in the sense that no subset of MDA is also a delaying alternative.

This branching scheme is illustrated in Figure 5.6. The root node corresponds to an empty schedule. Eligible activities are $\{A_1, A_3\}$, and they can be scheduled both at time 0 without violating resource constraints. The next node corresponds to the partial schedule and the relevant time-point is at time 2. Activity in progress is $IP(2) = \{A_3\}$, and $EL(2) = \{A_2, A_5\}$. If A_2 and A_5 are both added to the partial schedule then a resource conflict is detected. Thus, all minimal delaying alternatives, $MDA_l, l \in \{1, \ldots, 3\}$ are explicitly enumerated, and one node is created for each of them. The partial solution corresponding to MDA_l consists of scheduling $IP(t) \cup EL(t) \setminus MDA_l$ at time $t = 2$.

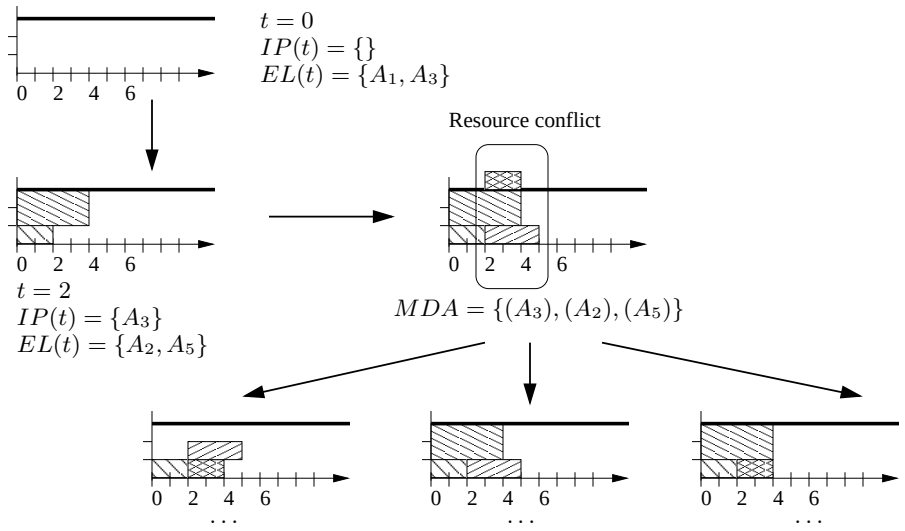


Figure 5.6. *Tree search based on delaying alternative scheme*

Related to this branching scheme, let us notice that Stinson et al. [STI 78], have proposed to enumerate *extension alternatives*: they consider subsets of activities that can be added to a partial solution without creating a resource conflict, whereas Demeulemeester and Herroelen [DEM 97] proposed to enumerate the set of activities that have to be delayed. One important point is that using these extension alternatives no scheduled activities are rescheduled at a lower level in the search tree, which may occurs using delaying alternatives. For instance, in the example of Figure 5.6, A_3 scheduled at the first node is removed from the partial solution if $MDA_l = \{A_3\}$ is the selected alternative. The main drawback of the extension alternatives is that the enumeration cannot be reduced to "maximal" extensions, due to the fact that a non-delay schedule may not be optimal, i.e., it may be interesting to delay an activity even if its resource requirements can be satisfied.

The following branching schemes that we present can be seen as an extension of the extension alternative concept.

5.1.3.2. *Building a solution using blocks*

The branching scheme proposed by Mingozzi *et al.* [MIN 98], does not build a feasible partial schedule at each node. Partial solutions may be made up of parts of activities: at each node of a search tree, a *block* (also called feasible set or feasible configuration; see Chapter 3) is added to the partial schedule, the duration of this block corresponds to the completion of one activity of the block, thus other activities may

be still in progress. A block is a set of activities that can be scheduled at the same time without violating neither the precedence constraints nor the resource constraints. The main interest of this approach is that blocks can be enumerated before the beginning of the construction of the tree, and thus reducing the time consumption of node treatment. Notice that even if this enumeration can be done before the exploration of the search tree, it may be very time-consuming due to the fact that the number of blocks is potentially exponential depending on the number of activities and their resource requirements. With each node N, corresponds to a time-point (the end of the previous scheduled block), a set of completed activities $SC(N)$, a set of in progress activities $IP(N)$ and a set of candidates blocks $FB(N)$. $FB(N)$ is made up of blocks whose activities have all their predecessors completed and contain in progress activities. One node is created for each candidate block.

Figure 5.7 shows a search tree based on this branching scheme. Let us consider the node presented at top of this figure. The set of completed activities is $SC(N) = \{A_1, A_3\}$, A_5 is still in progress at time 4, and all candidate blocks must contain A_5: $FB(N) = \{(A_5), (A_5, A_2), (A_5, A_4)\}$. One node is created corresponding to these candidates blocks. The time during which the block is scheduled corresponds to the smallest completion of one of the activities in progress. Only 1 time unit of A_5 is not scheduled at time 4, thus the candidate block will be scheduled during one time unit.

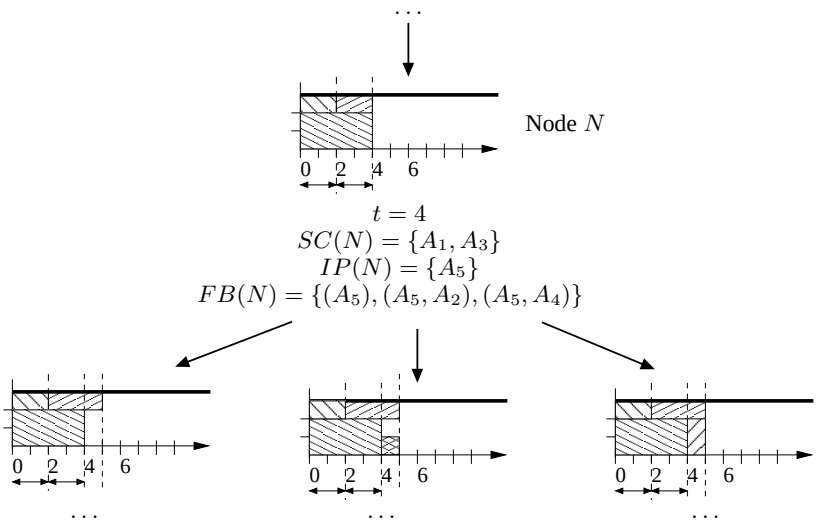


Figure 5.7. *Search tree built using block enumeration*

5.1.4. *Dominance rule: cut-set*

The two branching schemes presented so far generate partial schedules, so dominance rules presented in section 5.1.2 can be used to restrict the search to semi-active or active schedules. Moreover, Demeulemester and Herroelen [DEM 97] have proposed one of the most efficient dominance rules on classical instances defined in the literature.

This dominance rule is an *a posteriori* dominance rule. It is based on the comparison of two nodes, assuming that one of them is the current node and the second one was stored during previous step of the tree search. This comparison is based on the *cut-set* concept, $CS(t)$, defined at time t. It is the set of unscheduled activities for which all predecessors belong to the partial schedule. Let us consider two cut-sets $CS(t)$ corresponding to the current node (PS_t is the corresponding partial schedule) and $CS(t')$ previously stored ($PS_{t'}$ is the corresponding partial schedule). If $t' \leq t$ and if each activity A_j in progress at time t' does not finish in $PS_{t'}$ later than $\max(t, C_j)$, where C_j denotes the completion time of A_j in PS_t, then the current partial schedule PS_t is dominated by $PS_{t'}$. This dominance rule is presented in Figure 5.8.

This dominance rule is extremely efficient, and its use significantly reduces the size of the explored tree search. One of the drawbacks of this approach is the potentially huge number of cut-sets to store. However this approach makes it possible to solve to optimality medium size instances of traditional benchmarks (see Chapter 7).

5.2. Specific branching schemes

The branching schemes described in this section are slightly different than the previous ones: no partial schedules are built. Then specific dominance rules and lower bounds that are not based on this notion of partial schedules are defined.

5.2.1. *Fixing disjunction and parallel relationship*

This branching scheme was proposed by Brucker et al. [BRU 98]. It is an extension of a branching scheme proposed for solving job-shop problem. One node N is defined by 4 sets of activities. $C(N)$ is the set of pairs of activities that are linked by a precedence relation (also called a conjunction): $\{(A_i \leftarrow A_j)\}$. $D(N)$ is the set of pairs of activities that are in disjunction, i.e. that cannot be processed simultaneously, $\{(A_i - A_j)\}$. $\phi(N)$ is the set of pairs of activities that have to be processed simultaneously at least during one time-unit $\{(A_i||A_j)\}$, and finally $U(N)$ is the set of pairs of activities that are not in one of the three previous ones, $\{(A_i \sim A_j)\}$. At each node N, one element $(A_i \sim A_j)$ of $U(N)$ is chosen and two nodes are created: in the first

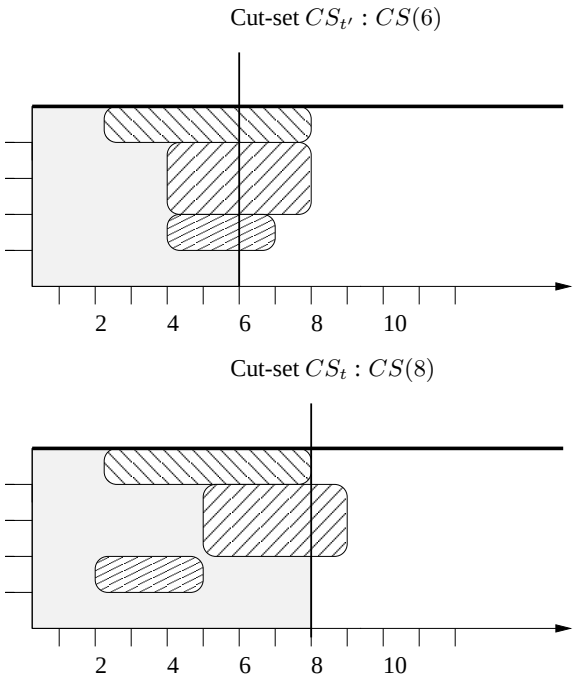


Figure 5.8. *Dominance rule based on cut-set*

node N_1, the two activities are enforced to be processed simultaneously at least during one time-unit, i.e., $\phi(N_1) = \phi(N) \cup \{(A_i||A_j)\}$ and $U(N_1) = U(N) \setminus \{(A_i \sim A_j)\}$, in the second node N_2 these two activities are enforced to be not processed simultaneously, i.e., $D(N_2) = D(N) \cup \{(A_i - A_j)\}$ and $U(N_1) = U(N) \setminus \{(A_i \sim A_j)\}$.

A leaf of such a tree is reached when $U(N) = \emptyset$, then a schedule having minimal makespan can be built in polynomial time.

One key point is how the pair of activities used to apply this branching scheme is chosen. The idea is quite similar to the one that is used in [CAR 91]: for all candidates pairs of activities in $U(N)$, lower bounds corresponding to N_1, and N_2 are computed, and then the pair of activities maximizing the sum of these two lower bounds is chosen.

Some mechanisms are used to deduce that some activities must or cannot be processed simultaneously and then add some pairs of activities either in $C(N)$ or $\phi(N)$, either based on distance matrix built using Floyd-Warshall algorithm or the concept of *symmetric triple of activities* [BRU 98]. Similar methods are related to CP approach and are presented in Chapter 4. This branching scheme is used in Chapter 12. The

schedule representation scheme based on the conjunction, disjunction and parallelity relations is also used in heuristic methods presented in Chapter 6.

5.2.2. *Reducing time-windows of activities*

Carlier and Latapie [CAR 91] have proposed an adaptation for solving the RCPSP of a branching scheme previously proposed for solving m-machine problem [CAR 87]. The main idea of this branching scheme, is to determine the set of feasible starting times of one activity defined as a critical one. Let UB_{best} denote the best known upper bound, $S_i \in \{ES_i, UB_{best} - q_i\}$, where $q_i = \mathcal{L}(A_i, A_{n+1})$, and $\mathcal{L}(A_i, A_j)$ denotes the length of the longest path in G from A_i to A_j. Once the set of feasible starting times are computed, then two nodes are created. In the first one the starting times of the chosen activity is restricted to the first half of the set of the feasible starting times, whereas in the second node it is restricted to the second half of this set, where m_i denotes the slack of A_i ($m_i = UB_{best} - q_i - ES_i - p_i$). In the first node N_1, we get $S_i \in \{ES_i, \ldots, ES_i + \lceil \frac{m_i}{2} \rceil\}$, and in the second node N_2: $S_i \in \{ES_i + \lceil \frac{m_i}{2} \rceil + 1, \ldots, UB_{best} - q_i - p_i\}$. This adjustment of earliest starting time and latest completion time are propagated to other activities using precedence constraints.

Thus, at each level of the search tree the slack of at least one activity is reduced, and a leaf of the search tree is reached when all slacks of activities are equal to 0. Then the feasibility of this set of starting-times can be checked polynomially.

To decide, at each node, the activity that is used for branching, i.e., the critical one, authors uses m-machine subproblem relaxation based lower bounds (LB) (see Chapter 2), and the activity that leads to the maximum increasing of the sum of lower bounds of the two nodes N_1 and N_2, i.e., $LB(N_1) + LB(N_2)$ is chosen.

This approach has two main drawbacks. First, activities are not fixed in the search tree (only the set of feasible starting times is reduced) and the depth of the search tree depends on the slack of activities, thus it is pseudo polynomial.

5.2.3. *Resolving forbidden sets*

As defined in section 1.6, a subset of activities F is called a forbidden set if there exists $R_k \in \mathcal{R}$ such that $\sum_{A_j \in F} b_{j,k} > B_k$. F is a *minimal forbidden set* if it is minimal with respect to the set inclusion. Thus at a given time point, the resource conflict, that corresponds to at least one forbidden set, can be resolved by adding precedence constraints between any two of its activities in order to delay one or more than one activity of the forbidden set.

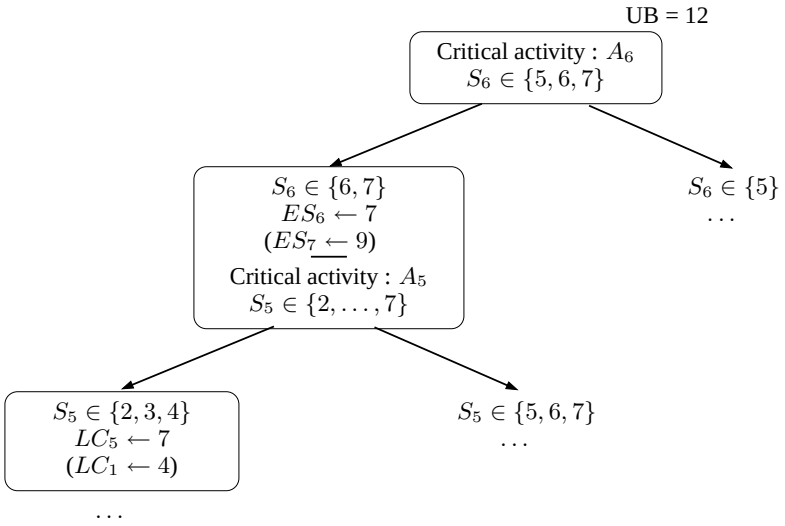


Figure 5.9. *Search tree based on time-window reduction*

Forbidden sets have been introduced by Ingelmund and Radermacher [IGE 83] for exact solving. Recently, theoretical aspects related to forbidden set enumeration have been investigated by Stork and Uetz [STO 05]. Laborie [LAB 05] presented a similar concept: *minimal critical sets*. More details about this method are presented in Chapter 7.

5.3. Conclusion and perspectives

In this chapter we have presented main branching schemes that are used to obtain exact solutions of the classical RCPSP. Notice that the efficiency of exact methods is obviously linked to the way that the search space is explored, i.e, to the branching scheme, but several works have proved that lower bounds and time-bounds reduction techniques are also extremely useful to improve the search, both in terms of number of explored nodes and time consumption. Thus, recent works that use cooperation between linear programming formulation and constraint programming are promising research directions.

Exact methods are used more and more to obtain efficient heuristic methods. First, the search tree can be truncated using, for instance, classical methods e.g., beam search or recovering beam search, or methods coming from the constraint programming community, e.g., limited discrepancy search. Exact methods are also used more and more as a part of heuristics based on large neighborhood search.

Chapter 6

Heuristics

This chapter describes the main heuristics proposed in the literature for the RCPSP. Special attention is paid to simple constructive heuristics and a generic schedule generation scheme embedding the standard serial and parallel schedule generations scheme is provided. For a complete description and experimental evaluation of standard heuristics and metaheuristics see [KOL 99a, KOL 06].

6.1. Schedule representation schemes

The fundamental goal of a heuristic algorithm is to generate a good solution in a reasonable amount of time. As such, it is necessary to define a representation scheme of the solution itself. In this section we describe the representation schemes used in the main heuristics of the literature. Other formulations or encoding solutions can be found in Chapters 3 and 4.

6.1.1. *Natural date variables*

Since we consider the makespan criterion in a nonpreemptive environment, the most straightforward formulation of a RCPSP chooses variables representing either the starting times or the completion times of activities. In fact, this representation is usually the final output of heuristics even when they operate on another internal encoding scheme. This formulation is in particular the core of the neighborhood procedure of Palpant *et al.* [PAL 04] described in section 6.4.4.

Chapter written by Christian ARTIGUES, David RIVREAU.

6.1.2. *List schedule generation scheme representations*

Typically, a heuristic process involves choosing between alternatives. A common practice to resolve this stage is to use a priority list as a decision maker. As a result the combination of the heuristic (schedule generator) and the priority list can be considered as solution provider for the RCPSP instance under investigation. In most cases the heuristic is supposed to be known in advance, so the list itself is seen as a coding for the solution. Due to its genericity and simplicity, the list representation is the basic solution encoding used by many greedy heuristics such as the serial and parallel scheduling schemes, but also by more elaborated heuristics such as tabu search or multi-pass methods (see section 6.2.1).

From the implementation point of view, the list either consists of activities in non increasing order of priority, or is a vector of priority values such that i-th element correspond to activity A_i. In the first case, we speak of an activity list whereas in the second the encoding is called random key representation. For instance, activity list (A_2, A_3, A_4, A_4) and random key vector $(4.6, 1.2, 1.9, 3.4)$ represent the same priority order in which activity A_2 is preferred to activity A_3, A_3 is preferred to A_4, and so on.

6.1.3. *Set based representations*

The minimal forbidden sets defined in Chapter 1 can also be used as the basis of an alternative encoding for solutions. We consider here that a forbidden set F can be defined as a set of activities such that there is no path between any pair of activities in the set and that there is a resource conflict on F. If no activity can be removed from that set without removing the conflict, then F is said to be a *minimal* forbidden set. Clearly, fixing $A_i \rightarrow A_j$ or $A_j \rightarrow A_i$ for $(A_i, A_j) \in F \times F$ solves the resource conflict related to F (and possibly some others). It follows that a solution to a RCPSP problem can be characterized by a selection of a set of at most one pair of activities for each minimal forbidden set such that all resource conflicts are resolved and there is no cycle in the augmented precedence graph. This is the strict-order representation defined in Chapter 1. Note that this representation is also called a partial-order schedule by some authors [GOD 05, POL 06].

Brucker *et al.* [BRU 98] devise a more sophisticated generation scheme representation (see also section 5.2.1). For these authors, a schedule scheme (C, D, ϕ, U) is a set of disjoint relations between activities. Relations in C are called conjunctions, D defines disjunctions and ϕ and U are parallelity and flexibility relations. More precisely a schedule scheme characterizes the set of solutions with:

– $A_i \rightarrow A_j$ for all $(A_i, A_j) \in C$ (A_i should precede A_j)

– $A_i - A_j$ for all $(A_i, A_j) \in D$ (A_i and A_j cannot be scheduled in parallel)

– $A_i||A_j$ for all $(A_i, A_j) \in \phi$ (A_i and A_j should be processed in parallel)

– $A_i \sim A_j$ for all $(A_i, A_j) \in U$ (there is no specific constraint between A_i and A_j)

When $U = \emptyset$, it can be proved that either no feasible solution satisfying all relations exists or a dominating feasible schedule can be found. So we can consider a schedule scheme of the form $(C, D, \phi, \emptyset)$ as a solution. A branching scheme using this representation is described in section 5.2.1 (see also section 12.3.1).

6.1.4. *Resource flow network representation*

Feasible schedules can finally be modeled by a network flow based representation (see Chapter 1. This formalization allows us to define a strict order that destroys all forbidden sets without explicitly enumerating all minimal forbidden sets. In this model the flow decision variables f_{ij}^k give the number of units of resource R_k directly transfered from an activity A_i to an activity A_j (where A_i belongs to $\mathcal{A} \cup \{A_0\}$, A_j is in $\mathcal{A} \cup \{A_{n+1}\}$ and R_k in $\mathcal{R}$). As shown in Chapter 1, from a feasible resource flow f, a schedule $ES(P(f))$ can be obtained through longest path computations in graph $G(V, E \cup P(f))$, where $P(f) = \{(A_i, A_j) \in V^2 | f_{ij} > 0\}$ and each arc is valuated by the duration of the origin activity. Conversely, a feasible flow can be built from any feasible schedule S by Algorithm 2.

6.2. Constructive heuristics

Constructive heuristics are often used to generate a first solution in very reasonable amount of time. In this section, we review the main methods applied for RCPSP.

6.2.1. *Standard list schedule generation scheme heuristics*

List schedules generation scheme (SGS) heuristics appear as one of the most widely used. Typically two basic scheduling schemes are reported: the serial scheduling scheme and the parallel scheduling scheme. Both heuristics schedule activities according to a given priority list, selecting tasks one by one and determining a starting time for each activity. Once a starting time is assigned to a particular task, this time is permanent and may not be changed at a later stage of the procedure.

Let $\mathcal{L}$ denote a list of activities and let $P(\mathcal{L})$ denote the strict (total) order induced by $\mathcal{L}$, i.e. $(A_i, A_j) \in P(\mathcal{L})$ if and only if A_i is before A_j in the list. $\mathcal{L}$ is said to be compatible with the precedence constraints whenever $E \subseteq P(\mathcal{L})$. Both the serial and the parallel scheduling schemes assume the priority list is compatible with the precedence constraints.

In the serial scheduling scheme proposed by Kelley [KEL 63], in each iteration the first available task in the list $\mathcal{L}$ is selected to be scheduled as soon as possible (with respect to precedence and resource constraints). Note the serial scheduling scheme is identical to the strict order agorithm proposed by Carlier (see [CAR 84] and Chapter 10).

On the other hand, the parallel scheduling scheme ([BRO 65]) operates in chronological fashion. At any period t, the set of *schedulable* activities, denoted by $\mathcal{L}_t$, consists of all those unscheduled activities which have their predecessors completed and which can be processed at that time with respect to resource constraints. Let $\underline{t}$ be the first time period such that $\mathcal{L}_{\underline{t}} \neq \emptyset$ at a given step of the procedure. Following a non-delay policy, the first activity in the priority list that belongs to $\mathcal{L}_{\underline{t}}$ is scheduled at time $\underline{t}$ and the process is iterated until all tasks are completed.

As pointed out by Debels *et al.* [DEB 06a], for both methods a single schedule can be represented by different priority lists. For instance, if we consider the activity list $(A_5, A_3, A_6, A_4, A_2, A_1)$ and that the corresponding solution start times are $S_5 = 0$, $S_3 = 2$, $S_6 = 0$, $S_4 = 5$, $S_2 = 8$, $S_1 = 7$, then activities A_3 and A_6 can clearly be swapped without changing the output. In order to eliminate this flaw, the authors propose to use a *standardized* random key representation for schedules. The central idea of Debels *et al.* consists of using rank values as priorities and applying a scheduling scheme to this random key distribution. This gives a starting time for each activity. The standardized random key representation is post-processed on this schedule by assigning the rank of the starting of activities as priorities. In case of ties, the lowest ranking is attributed to all activities starting at the same time. With this convention, each list corresponds to a unique schedule. For example, the standardized random key representation of previous schedule is $(5, 6, 3, 4, 1, 1)$ where priority values of A_5 and A_6 are equal to 1 since they have the same start time and they appear in first position.

Although serial and parallel scheduling schemes appear to be very similar at first sight (which is confirmed by the fact that they can be embedded into the generic framework of the next section), their output do not share the same properties. Indeed we will see that any schedule generated with the serial scheduling scheme belongs to the set of active schedules while only non-delay schedules are generated by the parallel scheduling scheme. From Sprecher *et al.* [SPR 95] it is also known that there is at least one optimal schedule that is active whereas the set of non-delay schedules may exclude all optimal solution[1]. These results are detailed in section 6.6.1.

1. Active and non-delay schedules are defined more precisely in Chapter 1.

From the operational point of view, list schedule generation scheme heuristics can be applied as single or multi-pass heuristics. A single pass generally consists of computing a list through a priority rule as described above. The list is said to be a static priority rule when it can be computed before running the scheduling algorithm. On the other hand, the dynamic priority rules are updated while the schedule is in construction. Typically, dynamic priority rules depend on the start time values. See [KOL 96] for a comprehensive survey of traditional priority rules for RCPSP.

Multi-pass heuristics are generally based on an iterative application of the serial or parallel schedule generation scheme. A different priority rule is used at each call, possibly by biasing the selection through a random device. Another variant of the multi-pass method is the iterative forward/backward scheduling method [LI 92a, ÖZD 96]. This method alternates forward and backward passes of the chosen schedule generation scheme. The forward pass is the standard application of the scheduling scheme while the backward pass applies the standard scheme to the mirror problem, obtained by reversing all arcs of the project network. In that way, the forward pass generates a left-shifted (or active) schedule while the backward pass generates a right-shifted schedule. Obviously, alternating forward and backward passes has an interest only if pertinent scheduling information is transmitted from each pass to the consecutive reverse one. Valls *et al.* [VAL 05] studied the performance of the *justification* technique. This technique, ensuring that the makespan cannot be increased between two consecutive passes [PAL 04], can also be defined as the neighborhood operator of a descent method and is detailed in section 6.2.4.

Forward and backward planning can be also integrated by computing in a single pass a schedule comprising a subset of activities scheduled by the forward schedule scheme while the remaining activities are scheduled by the backward scheme [KLE 00a]. At each step of this method, called bidirectional planning, the set of forward-scheduleable activities and the set of backward-scheduleable activities are computed according to the considered (serial or parallel) scheme. Then, if the activity selected for scheduling belongs to both sets, a priority rule is used to establish if the activity should be left-shifted or right-shifted.

6.2.2. *A Generic insertion-based list schedule generation scheme*

The purpose of this section is to show that the resource-flow representation can be used to embed both the parallel and the serial algorithms.

As seen in section 1.6, a complete schedule can be represented by a resource flow where the incoming and outgoing flows of any activity A_i on each resource R_k are equal to b_{ik} while the outgoing (respectively incoming) flow of dummy activity A_0 (resp. A_{n+1}) on each resource k is equal to B_k. We define a constructive algorithm initially setting $f^k_{0(n+1)}$ to B_k for each resource R_k and all other flows to 0, which

describes an empty schedule. Then, each activity is inserted in the schedule according to the order given by a list or priority rule. Each insertion assigns the incoming and outgoing resource flows of the inserted activity by always preserving the flow conservation property. The only restriction for such an insertion is that the augmented precedence graph $G(V, E \cup P(f))$ must remain acyclic. In such a framework, a way of determining the flow updates due to the insertion of an activity is to consider insertion positions. Before defining an insertion position, we need to introduce the concept of resource successors and predecessors of an activity.

DEFINITION 6.1.– *Given a resource flow f, the resource predecessor (resp. successor) of an activity A_i is the set of all activities A_j such that there is a resource R_k for which $f_{ji}^k > 0$ (resp. $f_{ij}^k > 0$)*

Now let A_x denote the activity to insert at a given step.

DEFINITION 6.2.– *An insertion position of an activity A_x in a flow f is a pair (α, β) of disjoint sets of already scheduled activities such that the set of resource predecessors of A_x is constrained to be a subset of α and the set of its resource successors is constrained to be a subset of β*

We define the concepts of valid and minimal insertion positions in order to obtain a feasible flow after insertion.

DEFINITION 6.3.– *A valid insertion position is an insertion position (α, β) such that there exists a feasible flow f' verifying $\sum_{i\in\alpha} f_{ix}^{'k} = \sum_{i\in\beta} f_{xi}^{'k} = b_{xk}$ on each resource $R_k \in \mathcal{R}$*

DEFINITION 6.4.– *A minimal insertion position is a valid insertion position (α, β) such that there exists a resource $R_k \in \mathcal{R}$ with $f_{ix}^{'k} > 0$ for each activity $A_i \in \alpha$ and a resource $R_{k'} \in \mathcal{R}$ with $f_{xi}^{'k'} > 0$ for each activity $A_i \in \beta$*

Note it may happen that $A_0 \in \alpha$. In the case where (α, β) is minimal, this indicates that A_x is inserted at the start of the schedule for a non-empty subset of resource units. Symmetrically, $A_{n+1} \in \beta$ means that A_x is inserted at the end of the schedule on a non-empty subset of resource units whenever (α, β) is minimal.

Once a flow f' is computed after insertion of an activity in a valid insertion position, the start times of all activities can be updated by longest path computations in the augmented precedence graph. A generic insertion-based schedule generation scheme is provided through Algorithm 4.

Different schedule generation schemes can be derived under this framework, depending on the way $\mathcal{L}$ is built, x^ρ and $(\alpha^\rho, \beta^\rho)$ are selected and the flow f is updated. We give hereafter the conditions of validity of an insertion position through the concepts of resource capacity and time intervals.

Algorithm 4 SGS($\mathcal{L}$): a generic list schedule generation scheme with input activity list $\mathcal{L}$

1: $f_{ij}^k \leftarrow 0, \forall A_i \in \mathcal{A} \cup \{A_0\}, \forall A_j \in \mathcal{A} \cup \{A_{n+1}\}, \forall R_k \in \mathcal{R}$.
2: $f_{0(n+1)}^k \leftarrow B_k, \forall R_k \in \mathcal{R}$.
3: **for** $\rho = 1, \ldots, |\mathcal{L}|$ **do**
4: Select an activity $A_{x^\rho} \in \mathcal{L}$ and remove it from $\mathcal{L}$
5: Compute a valid insertion position $(\alpha^\rho, \beta^\rho)$ for A_{x^ρ}.
6: Insert A_{x^ρ} into $(\alpha^\rho, \beta^\rho)$, update flow f and schedule S.
7: **end for**

DEFINITION 6.5.– *The resource capacity $b(\alpha, \beta)$ of an insertion position (α, β) is a $|\mathcal{R}|$-vector giving the number of resource units on each resource $R_k \in \mathcal{R}$ available for insertion of an activity in (α, β)*

DEFINITION 6.6.– *An insertion position (α, β) is valid for insertion of an activity x if all the following conditions are fulfilled.*

(1) There is no path in $G(V, E \cup P(f))$ from $A_i \in \beta \cup \{A_x\}$ to $A_j \in \alpha \cup \{A_x\}$, except the single node path $\{x\}$.

(2) $b(\alpha, \beta)_k \geq b_{ik}, \forall R_k \in \mathcal{R}$.

Condition (1) states that the insertion position has to satisfy the precedence constraints either set by precedence arcs E or by the current flow f. Condition (2) states that the insertion position has to provide a sufficient resource capacity. The way the capacities of the considered insertion positions are computed depends on the instantiation of the generic scheme.

The different components of the algorithm needed to obtain the standard serial and the parallel schedule generation scheme are described in sections 6.6.1 and 6.6.2 (Appendix). Special attention is paid on the implementation to obtain a time complexity as low as possible. However, the generic scheme based on the resource flows can be used into more complex insertion methods, possibly allowing the modifications of the start times of the inserted activities. Briand and Bezanger [BRI 06] have defined an *any-order* schedule generation scheme based on a polynomial insertion algorithm defined by Artigues and Roubellat [ART 00] and Artigues *et al.* [ART 03] (see also chapter 11). The any-order SGS differs from the serial and parallel SGS as its input activity list does not have to be compatible with the precedence constraints. The embedded insertion algorithm ensures that the precedence constraints are satisfied after each activity insertion.

6.2.3. *Set schedule generation scheme heuristics*

Alvarés-Valdez and Tamarit [ALV 89] (see also [SHA 65] and [BEL 91]) propose a heuristic based on the minimal forbidden sets representation. The idea is to add precedence relations between activities in order to suppress the forbidden sets. In each step of the procedure a minimal forbidden set is selected and then a new precedence relation is inserted between two activities in that set. Several strategies have been investigated. One of the most successful consists of the selection of a set with minimal cardinality and by adding the precedence relation that induces the minimum increase of makespan in the precedence graph. Algorithm 3 in Chapter 1 is precisely based on this principle. Such methods generate quasi-active schedules (see section 1.4) i.e. schedules in which no order-preserving left shift is feasible with respect to the strict-order represented by the schedule. This set of schedule is dominant but it is not the smallest dominant schedule set, as shown in chapter 1. In particular [SPR 95] show problem instances for which the schedules obtained by this framework are neither active nor semi-active.

As state in section 6.1.3, there is a polynomial algorithm to decode a solution from a schedule scheme of the form $(C, D, \phi, \emptyset)$. For the general case (C, D, ϕ, U), Baar *et al.* [BAA 98] devise a heuristic which aims to satisfy as many parallelity relations as possible.

6.2.4. *(Double-)justification-based methods*

As defined by Valls *et al.* [VAL 05], left (right) justifying an activity A_i in a schedule S amounts to finding the smallest (largest) start time for A_i leaving the start time of all other activities unchanged. For the right justification, an upper bound on the makespan is set. The left (right) justification of schedule S consists in left (right) justifying all activities A_i in the order of S_i (C_i). Palpant *et al.* [PAL 04] remarked that the left justification of schedule S cannot increase S_{n+1}. The reader may notice that the left justification of schedule S strictly amounts to applying the serial schedule scheme with a priority rule set to the start time of the activities in S, yielding an active schedule. Symetrically, let S' denote the right justification of schedule S. Palpant *et al.* [PAL 04] have shown that the schedule obtained by left shifting all activities in S' by S_0 is feasible and of makespan not greater than S_{n+1}. Starting from an initial schedule and performing a right justification then a left justification is called double-justification in [VAL 05]. More generally, alternating right and left justifications from an initial schedule until there is no more improvement of the makespan yields an iterated forward-backward scheduling procedure, as presented in section 6.2.1, and also a descent method. Valls *et al.* [VAL 05] and Palpant *et al.* [PAL 04] have shown the interest of embedding justification inside several metaheuristics.

6.3. Local search neighborhoods

Local search methods are iterative methods starting from a feasible initial solution, generally computed by a constructive heuristic, and applying at each iteration a local modification (also called move of neighborhood operator) to the current solution. Each particular local search method specifies the set of solutions $\mathcal{N}(x)$ that can be reached by a move from a solution x. $\mathcal{N}(x)$ is called the neighborhood of x. When designing a neighborhood, a desirable property is to ensure that there exists a sequence of moves to reach any solution from any other one. A neighborhood verifying this property is said to be strongly optimally connected. On the other hand, a neighborhood ensuring only that there exists a sequence of moves from any solution to the optimum is called weakly optimally connected. Another key point to consider in the definition of a neighborhood is its size, given by the maximal number of solutions in $\mathcal{N}(x)$. Increasing the size of $\mathcal{N}$ allows for exploring a larger search space at a single iteration, possibly reaching interesting solutions at the expense of larger computational requirements. This section describes neighborhoods for the RCPSP, each of them being closely related to the considered solution representation.

6.3.1. *List based neighborhoods*

The usual neighborhood operators have widely been used for the list based representations: relocate or shift [BOC 96, BAA 98, BOU 03, FLE 04], pairwise interchange [LEE 96], swap [PIN 94, KLE 00b]. Shifting an element in the list consists in selecting one element and moving it at another position. Pairwise interchange involves switching two adjacent elements in the list, whereas swap allows exchanging the positions of two elements even if they are not contiguous. All these neighborhood are strongly optimally connected, even if the considered list are restricted to be compatible with the precedence constraints, as it is generally required by the constructive scheduling algorithms (see section 6.2). Given a list of size n, the sizes of the shift and pairwise interchange neighborhoods are linear in n while the size of the swap neighborhood is quadratic in n.

Besides these basic moves, population based heuristics often generate new solutions from the combination of several selected individuals (see section 6.4.3). For instance, in a two-point crossover, two new candidate solutions are produced from the interchange of a selected subpart of a schedule between two parent lists[2]. Valls *et al.* [VAL 04, VAL 05] also propose several specific cross-operators for the random key representation. Finally, some high-level neighborhood strategies have been successfully applied on list representation: ant-colony optimization [MER 02], path-relinking method [KOC 03] and electromagnetism metaheuristic [DEB 06a].

2. A repair procedure is applied if necessary.

6.3.2. *Set based neighborhoods*

To the best of our knowledge, no improvement heuristic has been developed so far on the minimal forbidden set representation. That is not the case for the schedule scheme representation of section 6.1.3. Baar *et al.* [BAA 98] introduce four moves. The first one translates a flexibility relation into a parallelity relation. The second one eliminates all but one parallelity relations that are linked to a given activity A_i. The third move keeps a parallelity relation between two activities and removes the parallelity relations between these activities and the other ones. Finally, the last move suppresses all parallelity relations that belong to a connected parallelity component. For efficiency considerations, Baar *et al.* propose to restrict their investigation on an arbitrary subset of this $O(n^2)$-neighborhood.

6.3.3. *Resource flow based neighborhoods*

A move for the AON-flow network representation is a local modification of the flow yielding a different schedule. Artigues *et al.* [ART 03] propose as a move the removal of an activity from the flow network followed by its reinsertion at another position. The removal of an activity from the resource flow is done by an algorithm reassigning the flow between the former resource predecessors and successors of the activity. The reinsertion of the activity is performed by an insertion algorithm enumerating a polynomial number of insertion positions yielding a neighborhood size of $\mathcal{O}(n^3)$. The complexity is obtained by restricting the search, for a given activity to insert and a current flow f, to the insertion positions that keep the precedence constraints of $P(f)$. The insertion method is presented in Chapter 11 for an extension of the RCPSP.

The resource flow representation can be seen as an extension of the disjunctive-graph representation widely-used for the job-shop problem and its extensions. The standard neighborhoods defined for the disjunctive problems can be extended to the RCPSP, as proposed in [FOR 97]. To our knowledge, the connectivity of this class of neighborhoods has not yet been studied.

6.3.4. *Extended neighborhood for natural date variables*

A neighborhood $\mathcal{N}(S)$ of a schedule S with makespan $C_{\max}$ can be defined by considering all the feasible schedules of makespan not greater than $C_{\max}$ that can be obtained by fixing the start time of a subset of activities to their values in S. The size of this neighborhood is exponential in p, the number of unfixed activities, and its exploration defines an optimization subproblem that can be solved only for small sizes of p. Palpant *et al.* [PAL 04] reduce the search time by solving the exploration problem through constraint programming techniques (see Chapter 4). Godard *et al.*

[GOD 05], propose an extended neighborhood based on the strict order representation. The current schedule S is transformed into a strict order (which they call a partial-order schedule) by means of a polynomial algorithm. The objective of this transformation is to benefit from the flexibility introduced by the strict order, compared to the fixed start time schedule. Then, a subproblem is defined by the selection of a subset of activities and the removal from the strict order of all precedence constraints (except the structural ones) involving these activities. The neighborhood of the solution S is then defined as the set of solutions to the subproblem. Since the size of this neighborhood is exponential in the number of selected activities, a constraint programming solver is also used to select the best neighbor solution. Note that this precedence constraint-preserving neighborhood resembles the resource-flow based neighborhood defined in section 6.3.3, except that several activities are reinserted at a given step and that the resource flow model is not used to guide the search. To ensure diversification of the search, the set of activities defining the subproblem are defined for both methods through a random device.

6.4. Metaheuristics

In this section, we briefly review the various metaheuristic approaches, generally based on the representation and generation and neighborhood schemes described in the previous sections. The main purpose of this section is not to be really exhaustive but rather to show the variety of the proposed methods.

6.4.1. *Simulated annealing*

The simulated annealing belongs to the class of improvement heuristics. The core of the method is an attempt to simulate the way that liquids freeze and crystallize, or metals cool and anneal. According to the way the cooling process is carried out, i.e. slowly or quickly, the liquid metal is able to reach a minimal energy state (crystal form) or only a sub-optimal intermediate state. The idea of the simulated annealing method is based on an analogy of that physical process. At the beginning of the algorithm, an initial solution is chosen. Then a neighbor is created from that solution. If the neighbor has a better value than the current one it is accepted. If it does not improve the current solution, it may be accepted accordingly with a probability that is monotonically decreasing with E/T, where E is the absolute difference between solution values and T is the current temperature of the system. When the algorithm starts the temperature is high, so non improving solutions have a high probability to be accepted. As the process converges, the temperature is decreased, thus only small deteriorations of the objective value have a chance to be tolerated.

Boctor [BOC 96] and Bouleimen and Lecocq [BOU 03] propose to use the serial scheduling scheme on a priority list representation. The neighbor is generated using

the shift operator in the list. The Boulemein and Lecocq method appears to be one of the most efficient simulated annealing approaches to date [KOL 06].

Lee and Kim [LEE 96] and Cho and Kim [CHO 97] describe a simulated annealing approach based on the random key representation. They use the parallel scheduling scheme as a decoding procedure.

6.4.2. *Tabu search*

The tabu search is also an improvement heuristic that tolerates non-improving moves in the neihghbourhood. A short-term memory of recent moves or solutions called tabu list is used to prevent cycling between generated solutions.

Pinson *et al.* [PIN 94] were among the first to present an improvement heuristic based on activity list representation and the serial scheduling scheme. Three variations on the neighborhood have been investigated by Pinson *et al.*: pairwise interchange on the list, general swap and shift. It is noteworthy that this kind of procedure is still the backbone of many of the most successful heuristics at the moment. For instance, Kochetov and Stolyar [KOC 03] apply a evolutionary local search algorithm with variable neighborhood. As seen in section 6.3.1, the activity list is now helpfully replaced by improved representations: Valls *et al.* [VAL 03] use topological order representation in a tabu-like evolutionary metaheuristic when Debels *et al.* [DEB 06a] operate on standardized random key representation.

In parallel with their simulated annealing approach, Lee and Kim [LEE 96] also propose a tabu search on the combination of random key representation and parallel scheduling scheme. The neighborhood operator used is a restricted version of pairwise adjacent interchange.

Baar *et al.* [BAA 98] investigate two tabu search procedures. The first one is a variation of the serial scheduling scheme of the activity list representation that uses a specific shift operator which aims to cancel an arc that belongs to critical path. The second tabu search procedure of Baar *et al.* is based on the schedule scheme (C, D, N, F) and on the related neighborhood operators described in section 6.3.2.

Thomas and Salhi [THO 98] define a tabu search method which operates directly on natural date variables. They propose two different moves: activity swap that requires the starting times of two activities to be swapped and activity insert where the starting time of an activity is shifted to either the starting time or the completion time of another task. A repair procedure is applied to correct the infeasibilities induced on resources.

Finally, Artigues *et al.* [ART 03] propose a tabu search method using the neighborhood based on reinsertions in the resource flow network, defined in section 6.3.3.

The main interest of the resource flow network representation for local search is the concept of critical path associated with a flow f. A critical path is a longest path in graph $G(V, E \cup P(f))$. It is easy to see that the only modifications of the flow being able to improve the current schedule $ES(P(f))$ are the ones involving the activities located on this path. Thus in [ART 03], the reinsertions are limited to the critical activities.

6.4.3. *Population-based metaheuristics*

This section is devoted to evolutionary approaches in the broad sense: a population of solutions evolves accordingly to a specific algorithm. In general, at each step of the process, new candidate solutions are generated and a selection among individuals is applied to generate the new population. It should be noted that these methods appear to be very competitive for the RCPSP.

Hartmann [HAR 02] achieves substantial results with a genetic algorithm approach. An individual consists of an activity list and a selection of a scheduling scheme (serial or parallel). Hartmann uses two-point crossover on the activity lists and chooses to keep the mother's scheduling scheme for the daughter and the father's scheduling scheme for the son. Precedence-feasible shift-mutations are applied on lists and the scheduling scheme can also be reversed with a given probability.

Ant colony optimization together with local search has been used by Merkle *et al.* [MER 02]. In every generation, each of m ants builds a solution using a parallel or a serial scheduling scheme. The selection of activities for the scheduling algorithm is guided by a combination of pheromone information and heuristic evaluation.

Kochetov and Stolyar [KOC 03] create a hybrid evolutionary method based on path relinking, GRASP and tabu search (for an overview of these methods, see [GLO 97]). Solutions are represented by activity lists and an elaborated neighborhood is defined on starting time considerations. The main idea is to detect a set of resource concurrent activities. Each of these sets belongs to a sub-part somewhere in the middle of the activity list. The first and the last part of the list are scheduled according to a serial scheduling scheme. Activities in the second part are scheduled following an improved parallel scheduling scheme which aims to select on each step a subset with maximum resource utilization.

Valls *et al.* presents successful strategies in [VAL 03], [VAL 04] and [VAL 05]. All of these strategies involve the use of a population based on a basic version of the standardized random key representation. The various approaches of Valls *et al.* differ on the neighborhood moves or genetic operators used, the schedule generation scheme and the implementation of the evolution process.

As mentioned earlier, Debels *et al.* [DEB 06a] devise the Standardized Random Key representation and integrate an electromagnetism heuristic[3] based on that representation into a scatter search method[4]

6.4.4. *Additional methods*

Undoubtedly, there remain efficient methods that do not fall into the categories described so far.

For instance, Schirmer [SCH 00a] uses a case based reasoning adaptive search. The idea is to analyze the effect of instance characteristics to select the appropriate scheduling scheme/priority rule combination.

Möring *et al.* [MÖH 03] developed a Lagrangian based heuristic. Their Lagrangian approach relies on a time-indexed formulation in which the resource constraints are relaxed. The Langragian dual is solved using a subgradient procedure where the relaxed problem reduces to a minimum cut computation in a graph. Considering that the violations of the resource constraints tend to be reduced during subgradient optimization, and according to an idea inspired by approximation algorithms, Möring *et al.* propose to exploit the α-completion times of activities as input for the serial and parallel scheduling schemes.

Fkeszar and Hindi [FLE 04] implement the variable neighborhood search [HAN 99] metaheuristic upon the serial scheduling scheme with an improved shift move neighborhood.

[PAL 04] and [GOD 05] propose randomized large neighborhood search methods based on the extended neighborhoods presented in section 6.3.4. For these methods, randomization of the neighborhood definition prevents cycling and ensures diversification while the neighborhood large size allows intensification of the search. It follows that, by contrast to the other successful methods described above, there is no need of a complex metaheuristic embedding the neighborhood search. As a counterpart, sophisticated search methods are necessary to explore the neighborhood.

---

3. The electromagnetic approach, based on the theory of physics, simulates attraction and repulsion of sample points in order to move toward a promising solution.

4. Citing Glover [GLO 99]:

> Scatter search and path relinking contrast with other evolutionary procedures, such as genetic algorithms, by providing unifying principles for joining solutions based on generalized path constructions (in both Euclidean and neighborhood spaces) and by using strategic designs where other approaches resort to randomization.

Last, the exact methods described in Chapter 5 can be generally transformed into heuristics by interrupting the search before reaching optimality as soon as a feasible solution has been found.

6.5. Conclusion

According to the experimental evaluation of Kolisch and Hartmann [KOL 06], the best metaheuristics on the set of RCPSP instances proposed by Kolisch *et al.* [KOL 97] are almost all hybrid methods, incorporating several components, like evolutionary algorithms and local search. The emergence of these methods has been strongly motivated by a competition to obtain the best results on the KSD instances. Kolisch and Hartmann [KOL 06] conclude that this competition has the drawback to promote methods that associate existing results rather than test innovative ideas. Although it seems undeniable that hybrid methods are likely to obtain the best results on real problems, there remains an urgent need to focus on the essential and individual aspects of the heuristics. This would allow to understand in which cases some component should be used or not to solve particular problems. A small contribution of this chapter is to put a special emphasis on the schedule representation and generation schemes presenting a generic framework that could be used to propose new insertion-based constructive and local search algorithms, as also suggested in Chapter 11.

6.6. Appendix

In this appendix, we show that the serial and parallel scheduling schemes can be expressed inside the generic insertion-based scheduling framework.

6.6.1. *Serial list schedule generation revisited*

For the serial schedule generation scheme , the input activity list $\mathcal{L}$ has to be compatible with the precedence constraints such that no activity can be selected for insertion before any of its predecessors. The selected activity A_{i^ρ} is simply the ρ^{th} activity of list $\mathcal{L}$. As stated in section 6.2.1, the principle of the serial SGS is to schedule this activity as soon as possible without delaying any already scheduled activity. Given a partial schedule involving the set $\{A_{i^1}, \ldots, A_{i^{\rho-1}}\}$ of already inserted activities, we represent the insertion possibilities by means of a special structure: the event list $\mathcal{EL}$. An event e is 4-tuple $(t(e), \mathcal{S}(e), \mathcal{C}(e), (b_k(e))_{k\in\mathcal{R}})$:

– $t(e)$ denotes the time period of the event,

– $\mathcal{S}(e)$ is the set of activities starting exactly at time $t(e)$,

– $\mathcal{C}(e)$ is the set of activities completing exactly at time $t(e)$,

– $b_k(e)$ represents the remaining capacity of resource k available during interval $[t(e), t(\text{next}_e)[$ (next_e is the event immediately following e in $\mathcal{EL}$).

The event list is sorted in increasing order of $t(e)$ and all $t(e)$ are assumed to be distinct. It follows that each capacity $b_k(e)$ remains constant during interval $[t(e), t(\text{next}_e)[$. A partial or complete schedule can be represented by an event list $\mathcal{EL}$ and an empty schedule is denoted by $\mathcal{EL} = \emptyset$. To illustrate the concept of event list, consider the RCPSP example presented in Figure 6.1. Let us assume that activity $A_{i^\rho} = A_6$ has to be inserted as shown in the figure. We suppose here that activity A_6 has no predecessor. The event list $\mathcal{EL}$ of the partial schedule is given in Table 6.1.

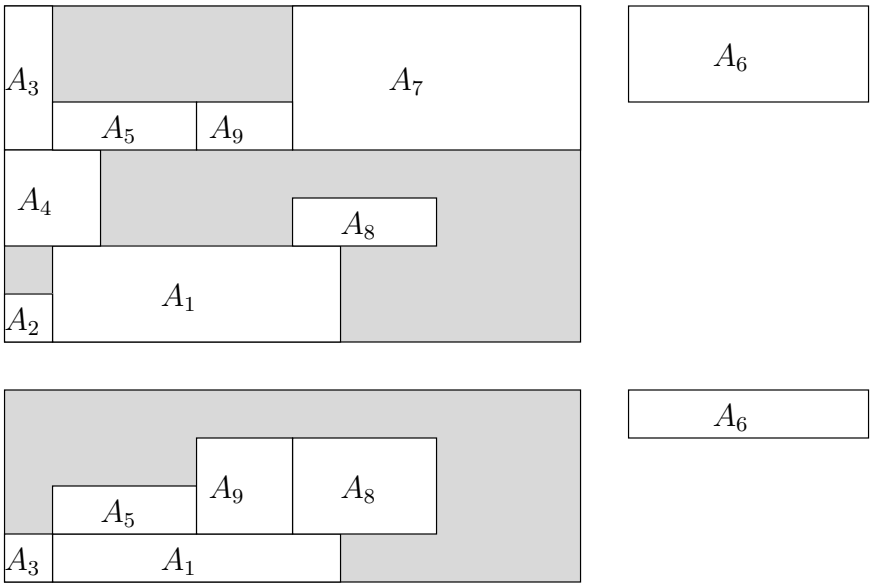


Figure 6.1. *Insertion of Activity A_6 by the serial scheduling scheme*

The insertion problem aims at finding the first event e such that the start time of activity A_{i^ρ} can be set to $S_{i^\rho} = t(e)$. Such an event is named a start event. The corresponding end event is the first event f after e in $\mathcal{EL}$ such that $t(e) + p_{i^\rho} \leq t(f)$. A_{i^ρ} is inserted between events e and f. Such events define unambiguously an insertion position (α, β) such that α is the set of activities completed exactly at or before $t(e)$, i.e. the unions of sets $\mathcal{C}(g)$ with $t(g) \leq t(e)$ plus activity A_0. β is the set of activities starting exactly at of after $t(f)$ i.e. the union of sets $\mathcal{S}(g)$ with $t(g) \geq t(f)$ plus activity A_{n+1}. The capacity of the insertion position is the minimal resource capacity available during each interval $[t(g), t(\text{next}_g)[$ where g varies from e to the event preceding f in $\mathcal{EL}$, denoted pred_f. In the example, the start event $e1 = (1, \{A_1, A_5\}, \{A_2, A_3\}, (2, 2))$

e	$t(e)$	$\mathcal{S}(e)$	$\mathcal{C}(e)$	$b_1(e)$	$b_2(e)$
$e1$	0	$\{A_2, A_3, A_4\}$	$\emptyset$	1	3
$e2$	1	$\{A_1, A_5\}$	$\{A_2, A_3\}$	2	2
$e3$	2	$\emptyset$	$\{A_4\}$	4	2
$e4$	4	$\{A_9\}$	$\{A_5\}$	4	1
$e5$	6	$\{A_7, A_8\}$	$\{A_9\}$	1	1
$e6$	7	$\emptyset$	$\{A_1\}$	3	2
$e7$	9	$\emptyset$	$\{A_8\}$	4	4
$e8$	12	$\emptyset$	$\{A_7\}$	7	4

Table 6.1. *Event list corresponding to the partial schedule of Figure 6.1*

and its end event $e5 = (6, \{A_7, A_8\}, \{A_9\}, (1, 1))$ define insertion position $(\alpha, \beta) = (\{A_0, A_2, A_3\}, \{A_7, A_8, A_10\})$. The capacity of this insertion position is the minimum capacity available among intervals $[t(e1) = 1, 2[$, $[2, 4[$, $[4, 6 = t(e5)[$, i.e. $b(\alpha, \beta)_1 = \min(2, 4, 4) = 2$ and $b(\alpha, \beta)_2 = \min(2, 2, 1) = 1$.

Algorithm 5 describes the serial schedule generation scheme, based on the event list structure. The method starts by assigning start time 0 to A_{i^1} and by initializing the event list with the start and completion events of A_{i^1} (steps 1 and 2). Then, for each activity A_{i^ρ}, $\rho > 1$ the first tentative start event e is set to the maximum completion times of the predecessors of A_{i^ρ} (step 4). The end event f is initially set to e and the algorithm then searches for the earliest feasible start and end events e and f for A_{i^ρ} (loop 6-13) stopping when the remaining duration μ, initially set to p_{i^ρ}, becomes null or when the end event is reached. Loop 6-13 works as follows. In case f is the last event of the event list, the remaining duration can be scheduled and e is a feasible start event. Otherwise, g denotes the event following f in the list and the capacity test is performed for the interval $[t(f), t(g)[$. If the test is positive, μ can be decreased by $t(g) - t(f)$ and f is set to g. Otherwise, the tentative start event e is unfeasible and e is set to g while μ is reset to p_{i^ρ}. The last steps (14-23) actually perform the insertion by setting the start time of A_{i^ρ} to $t(e)$ and updating the event list due to the insertion of A_{i^ρ}. The initialization takes $O(|\mathcal{R}|)$ time. The main loop has $n - 1$ iterations. Computing the initial start event e takes $O(n|\mathcal{R}|)$ time. The while loop enumerate all events in the worst case and each iteration is in $O(|\mathcal{R}|)$. Since there are less than $2n$ events, the while loop is in $O(n|\mathcal{R}|)$. The event list update takes also $O(n|\mathcal{R}|)$ time. The overall time complexity of the serial schedule generation scheme is consequently $O(n^2|\mathcal{R}|)$. From a computational point of view it is not necessary to update the flow at each iteration q. Indeed, the actual value of the flow is abstracted by the capacities $b_k(e)$. When a complete schedule is built, a feasible flow can be computed through Algorithm 2.

Algorithm 5 serialSGS($\{A_{i^1}, \ldots, A_{i^{|\mathcal{L}|}}\}$): the serial scheduling scheme

$S_{i^1} \leftarrow 0$
$\mathcal{EL} \leftarrow \{(0, \{A_{i^1}\}, \emptyset, (B_k - b_{i^1 k})_{R_k \in \mathcal{R}}), (p_{i^1}, \emptyset, \{A_{i^1}\}, (B_k)_{R_k \in \mathcal{R}})\}$
for $\rho = 2, \ldots, |\mathcal{L}|$ **do**
  Let $e \in \mathcal{EL}$ be the event such that $t(e) = \max_{A_j \in \Gamma_{i^\rho}^{-1}} C_j$
  $\mu = p_{i^\rho}$; $f \leftarrow e$;
  while f is not the last event of $\mathcal{EL}$ and $\mu > 0$ **do**
    $g \leftarrow \text{next}_f$
    if $b_k(f) \geq b_{i^\rho k}, \forall R_k \in \mathcal{R}$ **then**
      $\mu \leftarrow \max(0, \mu - t_g + t_f)$; $f \leftarrow g$
    else
      $\mu \leftarrow p_{i^\rho}$; $e \leftarrow g$; $f \leftarrow g$
    end if
  end while
  $S_{i^\rho} \leftarrow t(e)$
  insert A_{i^q} into $\mathcal{S}(e)$
  if $t(e) + p_{i^\rho} = t(f)$ **then**
    insert A_{i^ρ} into $\mathcal{C}(f)$
  else if $t(e) + p_{i^\rho} > t(f)$ **then**
    insert new event $g = (t(e) + p_{i^\rho}, \emptyset, \{A_{i^\rho}\}, (b_k(f))_{R_k \in \mathcal{R}})$ in $\mathcal{EL}$; $f \leftarrow g$
  else
    insert new event $g = (t(e) + p_{i^\rho}, \emptyset, \{A_{i^\rho}\}, (b_k(\text{pred}_f))_{R_k \in \mathcal{R}})$ in $\mathcal{EL}$; $f \leftarrow g$
  end if
  $b_k(g) \leftarrow b_k(g) - b_{i^\rho k}, \forall g$ between e and pred_f in $\mathcal{EL}, \forall R_k \in \mathcal{R}$
end for

As already shown by Kolisch [KOL 96], the obtained schedule S is an active schedule (see section 1.4). Indeed, at each iteration q, A_{i^ρ} cannot be scheduled earlier by construction. The serial scheduling scheme has also the following nice property. There exists a list $\mathcal{L}^*$ compatible with the precedence constraints such that applying the serial scheduling scheme with $\mathcal{L}^*$ yields an optimal schedule. This can be shown as follows. Consider an optimal schedule S^* and the activity list $\mathcal{L}$ obtained by sorting the activities in the increasing start time in S^*. A simple recursion argument shows that start time S_{i^ρ} issued by the serial algorithm at each step ρ verifies $S_{i^\rho} \leq S^*_{i^\rho}$.

6.6.2. *Parallel list schedule generation revisited*

For the parallel SGS, activity list $\mathcal{L}$ has also to be compatible with the precedence constraints. The principle is not to let a resource idle while it could be processing an activity, which yields a non-delay schedule (see section 1.4). Once an activity is inserted, its start time is also fixed until the end of the process. Algorithm 6 presents

the method using the sorted event list representation of the insertion positions. As for the serial SGS, the first activity of the list is scheduled at time 0 and its start and end events are created. The tentative start event e is set to the first event. At each iteration of the main loop (steps 5-27), the first activity A_i of list $\mathcal{L}$ that can be started at time $t(e)$ is considered: at step 6, $\pi(A_j)$ denotes the position of activity A_j in list $\mathcal{L}$. If such an activity A_i exists, it is removed from $\mathcal{L}$ and inserted in the position defined by e, updating the event list as for the serial scheme (steps 7-22). The earliest start times of its successors are updated (step 23). If A_i does not exist, no more activities can be scheduled at event e and the next event of $\mathcal{EL}$ is considered. The main loop has less than $3n$ iterations since for each of them, either an activity is scheduled or a new event is considered. At step 6, the search for A_i is in $O(n|\mathcal{R}|)$ and the update of the successor start times at step 23 takes only $|\Gamma_i|$ iterations. The remaining computations are similar to the ones of the serial SGS. It follows that the complexity of the parallel SGS is also $O(n^2|\mathcal{R}|)$. As underlined in section 6.2.1, since the parallel SGS generates only non-delay schedules there is no guarantee on the existence of a list $\mathcal{L}$ yielding an optimal solution.

Algorithm 6 ParallelSGS($\mathcal{L}$): the parallel schedule generation scheme

$A_i \leftarrow$first activity of $\mathcal{L}$; $\mathcal{L} \leftarrow \mathcal{L} \setminus \{A_i\}$
$\mathcal{EL} \leftarrow \{(0, \{A_i\}, \emptyset, (B_k - b_{ik})_{R_k \in \mathcal{R}}), (p_i, \emptyset, \{A_i\}, (B_k)_{R_k \in \mathcal{R}})\}$
$S_i \leftarrow 0, \forall A_i \in \{A_j \in V | \Gamma_j^{-1} = \emptyset\}$; $S_i \leftarrow \infty, \forall A_i \in \{A_j \in V | \Gamma_j^{-1} \neq \emptyset\}$
$e \leftarrow$ first event of $\mathcal{EL}$
while $\mathcal{L} \neq \emptyset$ **do**
  if $\exists A_i = \arg\min\{\pi(A_j) | A_j \in \mathcal{L}, S_j \leq t(e), b_k(e) \geq b_{jk}, \forall R_k \in \mathcal{R}\}$ **then**
    $S_i \leftarrow t$; $\mathcal{L} \leftarrow \mathcal{L} \setminus \{A_i\}$
    insert A_i into $\mathcal{S}(e)$
    $\mu \leftarrow p_i$
    $e \leftarrow f$
    while f is not the last event of $\mathcal{EL}$ and $\mu > 0$ **do**
      $\mu \leftarrow \max(0, \mu - t(\text{next}_f) + t(f))$
      $b_k(f) \leftarrow b_k(f) - b_{ik}, \forall R_k \in \mathcal{R}$
      $f \leftarrow next_f$
    end while
    if $t(e) + p_i = t(f)$ **then**
      insert A_i into $\mathcal{C}(f)$
    else if $t(e) + p_i > t(f)$ **then**
      insert new event $g = (t(e) + p_i, \emptyset, \{A_i\}, (b_k(f))_{R_k \in \mathcal{R}})$ in $\mathcal{EL}$; $f \leftarrow g$
    else
      insert new event $g = (t(e) + p_i, \emptyset, \{A_i\}, (b_k(\text{pred}_f))_{R_k \in \mathcal{R}})$ in $\mathcal{EL}$; $f \leftarrow g$
    end if
    $S_j \leftarrow \max(S_j, C_i), \forall j \in \Gamma_i$
  else
    $e \leftarrow \text{next}_e$
  end if
end while

Chapter 7

Benchmark Instance Indicators and Computational Comparison of Methods

7.1. Introduction

If there exist some strongly NP-hard problems that can be solved reasonably well in practice, the RCPSP belongs to the class of *really* hard optimization problems. To illustrate this, instances of the minimum lateness one-machine scheduling problem with release dates with thousands of jobs can be solved to optimality by Carlier's algorithm [CAR 82]. This is unfortunately not the case for the RCPSP for which today's exact methods are still not able to solve problems dealing with more than 60 activities.

A way to compare several exact and heuristic methods proposed to solve an NP-hard optimization problem involves comparing the results they obtain in terms of consumed CPU time, memory requirement and objective-function value on a common set of problem instances. Since the late 1960s, a wide variety of benchmark RCPSP instances has been proposed. The design of such RCPSP instances must take into account the following two major features that the RCPSP shares with many NP-hard problems.

First, the RCPSP has a complex structure and can be decomposed into many categories depending on the resource features, the activity requirements and the precedence-constraint graph. To give an idea of this diversity, simply consider the fact

---

Chapter written by Christian ARTIGUES, Oumar KONÉ, Pierre LOPEZ, Marcel MONGEAU, Emmanuel NÉRON, David RIVREAU.

that the job-shop scheduling problem (JSP) and the parallel machine problem (PMP) are both special cases of the RCPSP, both known for their hardness.

Second, as established by Herroelen and de Reyck [HER 99], the RCPSP admits phase transitions. Indeed, consider a set of RCPSP instances with a single resource ($|\mathcal{R}| = 1$), which allows us to drop the resource index in the notations. If B is larger than $\sum_{A_i \in \mathcal{A}} b_i$, then there is no resource constraint and the problem amounts to find feasible start times with respect to precedence constraints only, which can be performed in $O(n^2)$. The other extreme case is also trivial since if B is smaller than $\max_{A_i \in \mathcal{A}} r_i$, then there is no solution to the problem, which can be checked in $O(n)$. For the intermediary values, Herroelen and de Reyck showed in [HER 99] that the computational requirement of the branch-and-bound procedure developed by Demeulemeester and Herroelen [DEM 97] varies as a function of B according to a continuous bell-shaped easy-hard-easy complexity pattern.

The purpose of this chapter is twofold. First, we shall compare the performance of state-of-the-art methods for solving the RCPSP. Secondly, we shall evaluate the ability of several indicators to evaluate instance hardness. Indeed, an area of research is devoted to the proposition of various indicators able to foresee the hardness of a given instance. Section 7.2 is devoted to the presentation and the critical analysis of such indicators while section 7.3 proposes new instance indicators. The computational experiments carried out in this chapter are performed on a selection of representative instance subsets. section 7.4 refers to a presentation of the benchmark instances encountered in the literature. Section 7.5 analyzes the stability of the indicators and underlines some pitfalls, focusing on a particular set of instances. Section 7.6 first presents a sample of methods, also selected from the literature and described in the preceding chapters. Then, these methods are tested and compared on the selected instances under a common experimental framework. The results of these experiments are analyzed and conclusions are drawn. We conclude our chapter in section 7.7.

7.2. Standard instance indicators

Since the 1960s, indicators have emerged to characterize the RCPSP instances. They can be roughly classified into four categories: precedence-oriented, time-oriented, resource-oriented, and hybrid. In general, the correlation between an indicator and the instance hardness is established experimentally as follows. One uses an exact method to solve a series of groups of randomly generated instances with increasing indicator value and measures the average CPU time and its standard deviation for each group. In this section, we list a selection of state-of-the-art indicators from each category and recall the previously established relationships between the indicators value and the instance tractability.

We first consider the precedence-oriented indicators. Their objective is to evaluate the complexity of the resource network. We concentrate here on the main precedence-oriented indicators (recently, other indicators of this category were proposed in [VAN 04]).

OS: *order strength* [MAS 70]
The order strength (OS) can be defined as the density of the transitive closure TE of the precedence graph:

$$\text{OS} = |TE|/(n(n+1)/2).$$

We have $0 \leq \text{OS} \leq 1$, where $\text{OS} = 0$ corresponds to a total parallelism whereas $\text{OS} = 1$ means that the activities are totally ordered. It has been observed [HER 99] that the hardness of the problem instance decreases as the OS increases. Intuitively, the size of the search space decreases as precedence constraints are added to the network.

NC: *network complexity* [DAV 75]
The coefficient of network complexity (NC) corresponds to the average number of arcs per activity assuming E includes no redundant arcs:

$$\text{NC} = \sum_{A_i \in V} |\Gamma_i \cup \Gamma_i^{-1}|/2n.$$

Kolish *et al.* [KOL 95b] observed that the hardness of the instances decreases as NC increases, which can be explained with the same arguments as for the OS. However, some authors question the significance of this indicator. Elmaghraby and Heroellen [ELM 80] observed that instances with same NC may have completely different precedence graph structures. Moreover, de Reyck and Herroelen [de 96a] could not confirm experimentally the negative correlation between the NC and the instance hardness.

CI: *complexity index* [BEI 92]
The complexity index (CI) is used to determine how nearly series-parallel the precedence graph is. CI is defined as the minimum number of node reductions (along with series and parallel reductions) required to reduce the precedence graph G into a single arc graph. See [BEI 92] and [de 96a] for a more precise definition of the complexity index, the related node, series and parallel reductions, and for a description of an algorithm to compute the CI. De Reyck and Herroelen [de 96a] carried out a set of experiments showing that the instance hardness decreases as the CI increases. They explain this statement as the high complexity of the network restricts the possibility of scheduling activities in parallel, which makes the RCPSP instance easier. They also

conclude from their experiments that the CI alone is not sufficient to characterize the instance hardness since resource constraints play a major role.

The second category of indicators aims at defining the resource features.

RF: *resource factor* [PAS 66]
The resource factor (RF) is defined as the average number of required resources. Let $z_{ik} = 1$ if $b_{ik} > 0$, and $z_{ik} = 0$ otherwise. We have

$$\text{RF} = \frac{\sum_{A_i \in \mathcal{A}} z_{ik}}{n|\mathcal{R}|}.$$

It follows that $0 \leq \text{RF} \leq 1$. Kolisch *et al.* [KOL 95b] showed experimentally that the instance hardness increases as the RF increases. However, Herroelen [HER 06] remarked that the RF can be seriously biased is there is a significant number of activities without any resource requirement.

RC:*resource contrainedness* [PAT 76]
The resource contrainedness indicator measures the level of resource R_k availability compared to the average level of activity requirements for each resource $R_k \in \mathcal{R}$:

$$\text{RC}_k = \sum_{A_i \in \mathcal{A}} b_{ik}/nB_k.$$

De Reyck and Herroelen [de 96a] showed experimentally that the required CPU time varies as afunction of RC according to an easy-hard-easy bell curve. The extreme cases presented in the introduction of this chapter are intuitively consistent with this result. However, the RC indicator shares some drawbacks with the resource strength indicator RS, defined below.

The third category of indicators aims at defining the time features of an instance. We give one example of an indicator belonging to this category.

FFR: *free float ratio* [ALV 89]
The free float ratio is equal to:

$$\text{FFR} = \frac{\sum_{A_i \in \mathcal{A}} p_i}{\sum_{A_i \in \mathcal{A}} (p_i + FF_i)},$$

where FF_i denotes the free float of activity A_i:

$$FF_i = \min_{A_j \in \Gamma_i} ES_j - ES_i$$

and ES_i is defined as the earliest precedence-feasible start time of A_i without considering the resource availabilities. The free float of an activity describes its ability of being shifted without shifting any other activity in the absence of resource constraint. Thus, we have $0 \leq$ FFR ≤ 1. Alvarez-Valdés and Tamarit [ALV 89] call FFR: "density". They report from their experiments (on a limited set of instances) that instances with a low FFR are harder than the ones with a high FFR. Indeed the latter incorporate more flexibility, allowing the activities to be delayed without increasing the total completion time.

The last category of indicators gathers hybrid indicators mixing at least two of the above-defined categories. Thus, they have the desirable feature of providing a global characterization on the instance hardness.

RS:*resource strength* [KOL 95b]
The resource strength (RS) defines resource features incorporating also time features. It is defined for each resource $R_k \in \mathcal{R}$ as

$$RS_k = (B_k - b_k^{\min})(b_k^{\max} - b_k^{\min})$$

where $b_k^{\min} = \max_{A_i \in \mathcal{A}} b_{ik}$ is the largest activity requirement for R_k, and $b_k^{\max}$ is the peak demand for resource k in the precedence-based early start schedule ES:

$$b_k^{\max} = \max_{t \in \{0,\ldots,ES_{n+1}-1\}} \sum_{A_i \in \mathcal{A}_t} b_{ik}.$$

This indicator is designed so that the smallest feasible resource availability for a resource R_k corresponds to $\text{RS}_k = 0$ while $\text{RS}_k = 1$ corresponds to the absence of resource constraints since the peak resource demand for the earliest precedence feasible schedule is satisfied. Kolish *et al.* [KOL 95b] conjectured that the average solution time continuously decreases with RS. However, de Reyck and Herroelen [de 96a] performed experiments showing that the required CPU time varies as a function of RS according to a continuous bell-shaped easy-hard-easy pattern (with instances close to RS $= 0$ being far harder than the ones close to RS $= 1$). A major drawback of RS is that if an activity has a maximal demand on one resource, we have RS $= 0$ independently of the rest of the data. The RC indicator does not have this drawback. De Reyck and Herroelen [de 96a] also stated that, although instances close to RS $= 0$ (and consequently a high RC value) are on average hard to solve, there is a large variance for the CPU times. In other words, the RS and RC indicators are moderately significant. This can be partially explained by the ambiguity of these indicators: since they are defined for each resource, the value of RS (RC) for an instance is averaged over all resources. Thus, for resources with heterogenous strengths, the resulting RS and RC have only little significance.

DR: *disjunction ratio* [BAP 00]
The disjunction ratio (DR) integrates precedence and resource features:

$$\text{DR} = |TE \cup D|/(n(n+1)/2)$$

where TE is the transitive closure of the precedence graph, and D is the set of pairs of activities in disjunction: $D = \{(i,j)|\exists R_k \in \mathcal{R}, b_{ik} + b_{jk} > B_k\}$. Obviously we have DR $\leq$ OS and $0 \leq$ DR ≤ 1. Baptiste and Le Pape [BAP 00] defined this indicator to distinguish between cumulative instances (with a low disjunction ratio) and disjunctive instances (with a high disjunction ratio). The former are likely to be tackled efficiently by exploiting one-machine relaxations and disjunctive constraint propagation techniques (see sections 2.2 and 4.2.3). The latter may benefit from cumulative resource relaxations and energetic reasoning techniques (see sections 2.4 and 4.2.5). Baptiste and Le Pape [BAP 00] remarked that in the set of instances proposed by Kolish *et al.* [KOL 95b] (the KSD30 instances described in section 7.4), the hard instances (with small RS) all have a high disjunction ratio while all instances with a small disjunction ratio are easy. Thereby, they stressed the need to generate highly cumulative (hard) problem instances.

#FS and ACFS: *number and cardinality of minimal forbidden sets* [BAR 88, STO 05]
As stated in section 1.6, a minimal forbidden set is a set F of activities such that for some resource $R_k \in \mathcal{R}$, we have $\sum_{A_i \in F} b_{ik} > B_k$ and no subset $F' \subset F$ is forbidden. The number (#FS) and the average cardinality (ACFS) of minimal forbidden sets may be considered as indicators of instance hardness since, by definition, to obtain a feasible solution each minimal forbidden set F must be broken up by introducing at least one precedence constraint (i,j) with $i,j \in F$, which corresponds to $|F||F-1|$ possible decisions. With this definition, we obtain a pure resource indicator. Nevertheless, we can note (see [STO 05] and section 1.6) that some forbidden sets are implicitly broken up by precedence relations. These forbidden sets should not be taken into account to characterize the instance hardness. As a consequence, we define indicator #FS as the number of minimal forbidden sets that are also antichains in the transitively closed precedence graph $G(V, TE)$. Obviously the drawback of this indicator is that the number of minimal forbidden sets can be exponential in n. Bartusch *et al.* [BAR 88] designed a branch-and-bound algorithm whose performance decreases as the number of minimal forbidden sets and their cardinality increase. Stork and Uetz [STO 05] proposed recently an efficient algorithm to enumerate the minimal forbidden sets (in time polynomial in #FS) and they computed the #FS and ACFS indicators for the KSD30 instances. They showed that #FS and ACFS increase as the RF increases, which is consistent with the positive correlation established by Kolisch *et al.* [KOL 95b] between the RF and the instance hardness. However, they show that the number of minimal forbidden sets of the hardest instances is twice as low as the one of the easiest instances while the average minimal forbidden set cardinality is also significantly smaller. This counterintuitive (with respect to instance hardness) result indicates that the minimal forbidden set indicators are not sufficient

to characterize the hardness of an instance.

7.3. New instance indicators

We propose new indicators which allow us either to avoid drawbacks of the standard indicators or at least to illustrate and explain them better. We first introduce a new minimal forbidden set based indicator, then we define upper-bound based indicators.

#UFS and ACUFS: *number and cardinality of unsolved minimal forbidden sets*
We define the number of unsolved minimal forbidden sets (#UFS) as the number of minimal forbidden sets violated by the earliest precedence-feasible schedule ES. ACUFS denotes the average cardinality of such minimal forbidden sets. If the problem is trivial when #UFS=0, it intuitively may become harder as #UFS increases. However, it could also be that from a low #UFS, a single bad scheduling decision involves a significant increase of the number of violated FS or, conversely, that from a high #UFS, a single clever scheduling decision yields a significant decrease of #UFS.

+: *upper bound-based indicators*
To solve the RCPSP, we may add any valid makespan upper bound to the constraints without changing the objective-function value. It turns out that adding such a constraint may increase the number of precedence constraints. Indeed, consider a RCPSP instance I1 with 4 non-dummy activities of durations $p_1 = 2$, $p_2 = 1$, $p_3 = 2$, $p_4 = 2$ linked by precedence constraints (A_1, A_2) and (A_3, A_4). If we set a makespan upper bound to 4, then A_3 must precede A_2. This is due to the fact that setting the upper bound creates an arc from A_5 to A_0 valuated by -4 and also a path of length $2 = p_3$ from A_3 to A_2 going through this arc. Removing this arc and setting the precedence constraint (A_3, A_2) yields an RCPSP instance I2 with the same optimal makespan and with a solution space that is a strict subset of that of I1.

Given a valid upper bound UB for an instance I, all implied precedence relations can be computed in $O(n^3)$ via the Floyd-Warshall algorithm which computes the longest paths between every pair of activities. Let $ubf(UB, I)$ denote the instance obtained by the upper-bounding transformation of instance I. It seems reasonable to consider that adding such upper-bound implied precedence constraint should not increase the hardness of the instance. Moreover, $ubf(UB, I)$ and I should remain close to each other with respect to any hardness indicator.

More formally, assume that an indicator X is positively correlated with the instance hardness. We define the relation $\overset{X}{\prec}$ between instances where $I1 \overset{X}{\prec} I2$

states that instance $I1$ is *significantly* harder than instance $I2$ according to indicator X. Then, the following are two reasonable properties that an indicator X should satisfy. First, $I \overset{X}{\preceq} ubf(UB, I)$ holds since adding upper-bound implied precedence constraint should not increase the hardness of the problem. Second, if $I1 \overset{X}{\prec} I2$, $ubf(UB1, I1) \overset{X}{\prec} ubf(UB2, I2)$ where $UB1$ and $UB2$ are valid upper bounds for $I1$ and $I2$, computed with an equivalent computational effort.

In order to obtain an upper bound for each RCPSP instance considered in section 7.4, we apply a simple priority-rule based constructive algorithm, namely the parallel scheduling algorithm (see sections 6.2.1 and 6.6.2) with the MINLFT priority rule. This priority rule selects the activity A_i with the minimum latest finishing time LC_i in the latest precedence-feasible schedule given an arbitrary upper bound (see section 2.1).

Adding new precedence constraints through this transformation possibly modifies indicators OS, NC, CI, DR, #FS and ACFS. We denote by OS+, NC+, CI+, #FS+ and ACFS+ the value of the indicators after the transformation. However, it can be shown that indicators RC, RF, FFR, RS, #UFS and ACUFS are not changed by this transformation. This can be interpreted as a nice feature of the latter indicators.

7.4. State-of-the-art benchmark instances

In this section we describe the most studied RCPSP instances, some of which have been incorporated in our computational studies (sections 7.5 and 7.6).

A set of 110 RCPSP instances collected by Patterson [PAT 84] was widely used as the only benchmark RCPSP instance set until 1995. However, as underlined by Kolish *et al.* [KOL 99c], this instance set has the major drawback of not being designed according to well-specified parameters. Moreover in 1992, Demeulemeester and Herroelen [DEM 92b] proposed a branch-and-bound procedure (named DH) able to solve exactly all the Patterson instances with an average (maximal) CPU time of 0.76 (14) seconds on a personal computer (IBM PS 2 model 55sx - 80386sx processor running at 15Mhz with 640 KB RAM).

On the basis of the indicators proposed by Cooper [COO 76], other problem instances were generated by Lawrence [LAW 84] and later by Alvarez-Valdés and Tamarit [ALV 89]. With this generator each indicator becomes a parameter for instance generation. More precisely, Alvarez-Valdés and Tamarit [ALV 89] generated 144 instances, each instance corresponding to a fixed parameter (indicator) setting: $n \in \{25, 49, 101\}$, NC $\in \{1, 2, 3\}$, FFR $\in \{0.5, 0.75\}$, RF $\in \{0.5, 1\}$, RC $\in \{1/5, 1/4, 1/3, 1/2\}$. Duration and demands are randomly generated in the interval $[1, 10]$ and the resource availabilities are computed according to the RC. A

drawback of their instance set is that they propose a single instance per combination of indicators. This is restrictive for establishing correlations experimentally between indicator values and instance hardness. Alvarez-Valdés and Tamarit [ALV 89] report that all the 27-activity and almost all the 51-activity instances (named ALV27 and ALV51) were solved by one of the branch-and-bound algorithms of [CHR 87] and [STI 78]. Beldiceanu *et al.* [BEL 96] solved to optimality 33 out of 48 instances with 103 activities (named ALV103) using the constraint programming solver CHIP. The authors report that for one third of the solved instances, the optimality proof could require two weeks of CPU time. Baptiste [BAP 98] reports that the average disjunction ratio for the ALV27 and ALV51 instances is high (0.83). He solved via backtrack search 80 out of the 96 ALV27 and ALV51 instances within 1800s on a PC Dell OptiPlex GX Pro running at 200 Mhz under Windows NT with an average CPU time of 67.7 seconds.

Kolish, Sprecher and Drexl generalized Alvarez-Valdés and Tamarit approach by designing a problem generator (named PROGEN) based on the NC, RF, and RS indicators [KOL 95b]. With PROGEN, they generated today's most used RCPSP instances, gathered in the PSPLIB project scheduling library. The instances are publicly available through the web site [PSPib] and are described in [KOL 97, KOL 99c]. These instances were generated in groups of 10 instances, each of which has approximatively common parameter values (NC, RF and RS). There are 48 groups of instances with $n = 30$ activities (named KSD30), 48 groups of instances with $n = 60$ activities (named KSD60), 48 groups of instances with $n = 90$ activities (named KSD90) and 60 groups of instances with $n = 120$ activities (named KSD120), yielding a total number of 2,040 instances. The 48 groups of instances with 30, 60 and 90 activities are obtained by taking every combination of NC $\in \{1.5, 1.8, 2.1\}$, RF $\in \{0.25, 0.50, 0.75, 1.0\}$ and RS $\in \{0.2, 0.5, 0.7, 1.0\}$. The 60 groups of instances with 120 activities are obtained by the same range of NC and RF values but combined with RS $\in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. In 1995, 479 of the 480 KSD30 instances were solved to optimality by a new version (see section 5.1.3.1) of the branch-and-bound procedure of Demeulemeester and Herroelen [DEM 92b] (DH) with an average CPU time of 14.76 seconds. The remaining instance was solved within 3 hours (on an IBM PS2 Model P75 with a 80486 processor running at 25 Mhz and 24 MB RAM) [DEM 97]. Kolish *et al.* [KOL 95b] showed that the hardest instances of this set (for which the DH procedure required the largest CPU times) have the smallest resource strength (RS=0.2). As already reported in section 7.2, Baptiste and Le Pape [BAP 00] remarked that these instances have a high disjunction ratio (0.65 on average). The KSD60, KSD90 and KSD120 instances are still not all solved to optimality. The best known results to date for these open problems were obtained by Laborie [LAB 05] who closed 85% of the KSD60 instances, 79.4% of the KSD90 instances and 41.7% of the KSD 120 instances within a maximal CPU time of 1,800s per instance with the complete MCS-based search procedure (on a DellLatitude D600 laptop running at 1.4 Ghz). Tables 7.1 and 7.2 give a summary of

the indicators for the 480 KSD30 and 480 KSD60 instances. The average, minimal and maximal values as well as the standard deviation are displayed. We also give the proportion of instances with no unsolved forbidden sets, and the proportion of instances solved by the heuristic used to compute the upper bound, which happens when the upper bound is equal to the critical-path lower bound.

	$n+2$	$\|\mathcal{R}\|$	OS	NC	RF	RS	DR	#FS	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	32	4	0.52	1.81	0.63	0.62	0.56	326.24	3.45	17.01	2.2	0.58	2.15	0.61	241.77	3.44
Min	32	4	0.34	1.5	0.25	0.14	0.36	6	2.02	0	0	0.34	1.5	0.4	2	2.02
Max	32	4	0.69	2.13	1	1	0.9	4411	7.38	159	6.18	0.81	3.25	0.9	3232	7.73
σ	0	0	0.09	0.26	0.28	0.29	0.11	508.33	1.02	21.92	1.44	0.09	0.32	0.09	360.84	1.02

% instances with #UFS $= 0$: 25%, % instances trivially solved: 35.2%

Table 7.1. *Indicators for the 480 KSD30 instances*

	$n+2$	$\|\mathcal{R}\|$	OS	NC	RF	RS	DR	#FS
Avg	62	4	0.4	1.81	0.63	0.6	0.41	360,535
Min	62	4	0.25	1.5	0.25	0.14	0.25	69
Max	62	4	0.57	2.11	1	1	0.68	19,723,866
σ	0	0	0.08	0.25	0.28	0.29	0.09	1,676,354

	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	5.21	124.52	3.06	0.49	2.72	0.5	128,953	5.17
Min	2.38	0	0	0.27	1.5	0.27	31	2.38
Max	12.01	1,958	10.34	0.7	4.19	0.7	10,338,140	12.54
σ	1.9	227.28	2.16	0.1	0.62	0.09	646,581	1.9

% instances with #UFS $= 0$: 25%, % instances trivially solved: 38.7%

Table 7.2. *Indicators for the 480 KSD60 instances*

In Tables 7.1 and 7.2 we emphasized the values of the indicators NC, RF and RS used as generation parameters for the instances. As already mentioned, the instances have a high DR (more than 0.5 in average for the KSD30 and more than 0.4 on average for the KSD60) which is also a consequence of a high average OS. The upper-bounding procedure has a significant impact on the OS, NC, #FS and ACFS indicators (as shown by the "+"-superscripted indicators). The stability of these indicators will be specifically studied in section 7.5. For the KSD30 instances there is a reasonable average and maximum numbers of minimal forbidden sets (326 and 4,411, respectively) as already computed in [STO 05]. The upper-bounding procedure significantly reduced the average and maximum #FS (241 and 3,232, respectively). For the KSD60 instances, the average and maximum #FS increase drastically to 360,535 and almost 20 million, respectively. The upper-bound based number of minimal forbidden set is much more smaller with an average value of 128,953 and a maximum of about 10 millions. We thereby emphasize that taking account of the upper bound in

the #FS computation is crucial for the efficiency of any solution procedure based on the enumeration of the minimal forbidden sets. For the KSD30 instances, the average cardinality of the minimal forbidden sets ranges from 2 activities to 7 activities with an average value of 3.45 activities. The upper bounding procedure does not change these values significantly. The number of minimal forbidden sets unsolved by the earliest precedence-feasible schedule (#UFS) is much smaller than the total #FS. For the KSD30, it ranges from 0 to 159 with an average of 19. For the KSD60, it varies from 0 to 1958 with an average of 124. Note again that this indicator does not depends on the upper bound. The average cardinality of the unsolved minimal forbidden sets (ACUFS) is also smaller with an average of 2.2 instead of 3.45 for the ACFS. However, there are instances where the average cardinality of unsolved minimal forbidden sets is high, as shown by the maximum ACUFS of more than 6 activities for the KSD30 and more than 10 activities for the KSD60. Last, we observe that a large portion of the KSD30 and the KSD60 instances have no unsolved minimal forbidden sets (25% in both cases), corresponding to the instances having no resource constraints (RS=1). A significantly larger number of KSD30 and KSD60 instances (35.2% and 38.7%, respectively) are trivial since the simple upper-bounding heuristic is able to find a feasible solution with a makespan equal to the critical-path lower bound.

Baptiste and Le Pape [BAP 00] proposed a set of 39 instances (named BL), among which 19 comprise 20 activities and 20 involve 25 activities. Each activity requires 3 resources with a demand randomly generated between 0 and 60% of the availability. For the 20-activity instances, 15 precedence constraints were randomly generated while 45 precedence constraints were generated for the 25-activity instances. It follows that the average disjunction ratio is kept low (0.34 on average). The authors observed that the BL instances could not be solved by their backtrack search algorithm unless it integrates cumulative resource filtering algorithms, such as energetic reasoning presented in section 4.2.5. The instances were all solved to optimality within 1,800 seconds (on a PC Dell OptiPlex GX Pro running at 200 Mhz under Windows NT) with an average CPU time of 38.4 seconds. The indicator values for the BL instances are displayed in Table 7.3.

	$n+2$	$\|\mathcal{R}\|$	OS	NC	RF	RS	DR	#FS	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	24.56	3	0.34	1.67	0.66	0.34	0.34	526.87	3.53	86.59	3.37	0.34	1.72	0.34	514.03	3.53
Min	22	3	0.25	1.45	0.5	0.16	0.25	217	3.06	15	3	0.25	1.45	0.25	210	3.06
Max	27	3	0.45	2	0.77	0.55	0.45	1042	4.17	174	4.34	0.47	2.15	0.47	1042	4.14
σ	2.53	0	0.07	0.13	0.06	0.09	0.07	186.11	0.24	41.99	0.26	0.07	0.16	0.07	182.14	0.24

% instances with #UFS $= 0$: 0%, % instances trivially solved: 0%

Table 7.3. *Indicators for the BL instances*

Although the average disjunction ratio is low, the RS is also smaller on average (0.34) than for the KSD instances (0.56 and 0.41). Although the BL instances contain

fewer activities than the KSD30 ones, the average number of minimal forbidden sets is significantly larger (526 vs. 326). The average cardinality of the minimal forbidden sets is comparable to that of KSD30 while the maximal ACFS remains lower. The number of unsolved minimal forbidden sets is also significantly larger than for KSD30 (86 vs. 17). The upper bound has much less impact than for the KSD instances for all indicators. This is a consequence of the large gap between the critical-path lower bound and the upper bound. In particular the reduction from #FS to #FS+ is marginal. Also, there are no instances with #UCFS=0 and no instance was provably solved by the upper-bounding procedure.

Carlier and Néron [CAR 03] proposed a set of 55 instances (named Pack) with a small number of precedence relations. The number of activities varies from 17 to 35 (with an average of 25 activities) and there are 3 resources. Resource availability ranges from 5 to 10. There are two instance categories. In the first, the activity demand is randomly generated between 0 and B_k which may generate disjunctions. In the second, any activity demand cannot exceed half of the resource availability, which implies the absence of disjunctions and consequently yields highly cumulative instances. They solved 80% of the instances within 1000s with an average CPU time of 476s by using a branch-and-bound algorithm incorporating redundant cumulative resource generation and energetic reasoning (see sections 2.4 and 4.2.5). The indicator values for the Pack instances are displayed in Table 7.4.

	$n+2$	$\lvert\mathcal{R}\rvert$	OS	NC	RF	RS	DR	#FS	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	25.07	2.85	0.23	1.61	1	0.17	0.44	1426.93	3.23	481.76	3.13	0.23	1.61	0.44	1425.84	3.23
Min	17	2	0.13	1.5	1	0.08	0.19	115	2	18	2	0.13	1.5	0.19	115	2
Max	35	5	0.48	1.72	1	0.53	0.94	5774	4.02	2399	4	0.48	1.72	0.94	5774	4.02
σ	5.1	0.49	0.07	0.06	0	0.08	0.19	1623.06	0.47	557.86	0.49	0.07	0.06	0.19	1623.28	0.47

% instances with #UFS $= 0$: 0%, % instances trivially solved: 0%

Table 7.4. *Indicators for the 55 Pack instances*

As mentioned earlier, the table exhibits low average OS and NC values compared to the KSD and BL sets, a systematic high RF (each resource is required by each activity), a low RS, and an equally distributed DR from higly disjunctive to highly cumulative instances. All indicators are rather insensitive to the upper-bounding procedure, a consequence again of the large gap between the critical-path lower bound and the upper bound. No trivial instances have been detected. The average #FS is especially high (2.7 times larger than for BL and 4.3 times larger than for KSD30), as well as the average #UFS (5.5 times larger and 28 times larger respectively). The ACFS and ACUFS are of the same order as for BL and KSD30.

7.5. A critical analysis of the instance indicators

We take the KSD60 instances as a case study to analyze the discriminating power of the indicators using the information we have so far about trivial and non-trivial instances, to study the indicators stability with respect to the upper-bounding procedure and to find the correlations between indicators. The correlation between the instance hardness and the indicators will be further studied through solution method comparisons in section 7.6.

7.5.1. *Indicator comparison between trivial and non-trivial instances*

As noted in the preceding section, a significantly large number of KSD30 and KSD60 instances (35.2% and 38.7%, respectively) are trivial since they either have no unsolved minimal forbidden sets or the simple upper-bounding heuristic is able to find a feasible solution with a makespan equal to the critical-path lower bound. Removing these trivial instances, we obtain 311 KSD30 and 294 KSD60 instances that we refer to as non-trivial. The values of their indicators are displayed in Tables 7.5 and 7.6, respectively. When the trivial instances are well distributed among the possible values of an indicator, the average indicator value does not vary noticeably after the instance removal. This is the case for OS, NC, RF, DR, OS+ and DR+. On the other hand, besides the obvious increase of #UFS and #ACUFS, the remaining indicators show an important increase or decrease: NC+ (-4%,-9%), #FS (-6%,-76%), ACFS (-10%,-16%), #FS+ (+6%,-64%), ACFS+ (-10%,-16%), RS (-25%,-28%). The RS decreases since all instances such that RS=1 are removed.

	$n+2$	$\|\mathcal{R}\|$	OS	NC	RF	RS	DR	#FS	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	32	4	0.52	1.81	0.63	0.46	0.58	306.15	3.09	24.21	2.86	0.55	2.06	0.61	257.09	3.08
Min	32	4	0.34	1.5	0.25	0.14	0.36	8	2.02	2	2	0.34	1.5	0.43	8	2.02
Max	32	4	0.69	2.13	1	0.89	0.9	2,840	5.5	159	5.75	0.74	2.97	0.9	2,054	5.54
σ	0	0	0.09	0.26	0.27	0.21	0.11	418.32	0.7	23.41	0.71	0.09	0.31	0.1	322.07	0.7

Table 7.5. *Indicators for the 311 non-trivial KSD30 instances*

In order to have a first indication on the possible correlation between the indicators and the instance difficulty, we may consider the indicator values for the trivial instances such that #UFS$\neq$0, i.e. the instances easily solved by the constructive heuristic but having a strictly positive number of unsolved minimal forbidden sets. This involves 49 KSD30 instances and 66 KSD60 instances. The indicator values for these instances are given in Tables 7.7 and 7.8.

Compared to the non-trivial instances, the trivial ones with unsolved minimal forbidden sets have higher NC+ (+8%,+25%), #FS (+57%,+875%), ACFS

	$n+2$	$\|\mathcal{R}\|$	OS	NC	RF	RS	DR	#FS
Avg	62	4	0.4	1.81	0.62	0.43	0.42	84,673.53
Min	62	4	0.25	1.5	0.25	0.14	0.25	130
Max	62	4	0.57	2.11	1	0.76	0.68	3,774,139
σ	0	0	0.08	0.25	0.28	0.2	0.09	312,865.6

	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	4.37	175.63	3.81	0.46	2.46	0.48	46,072.6	4.34
Min	2.38	5	2	0.27	1.5	0.27	52	2.38
Max	8.74	1958	7.42	0.66	3.9	0.68	1,609,652	8.71
σ	1.26	250.97	1.22	0.09	0.59	0.09	157,472.18	1.25

Table 7.6. *Indicators for the 294 non-trivial KSD60 instances*

	$n+2$	$\|\mathcal{R}\|$	OS	NC	RF	RS	DR	#FS	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	32	4	0.51	1.81	0.61	0.66	0.52	483.14	3.62	12.96	3.35	0.6	2.23	0.6	306.08	3.58
Min	32	4	0.37	1.5	0.25	0.35	0.37	14	2.14	2	2	0.44	1.75	0.43	10	2.12
Max	32	4	0.67	2.13	1	0.82	0.69	3413	6.4	55	6.18	0.74	2.81	0.75	1823	6.41
σ	0	0	0.08	0.26	0.32	0.1	0.09	721.28	0.99	13.4	1.01	0.07	0.26	0.08	454.67	1

Table 7.7. *Indicators for 49 trivial KSD30 instances with unsolved minimal forbidden sets*

(+17%,+35%), #FS+(+19%,+756%), ACFS+(+16%;+34%), RS (+43%,+55%), ACUFS (+17%,+38%) whereas they have a lower #UFS (-46%,-29%). Thus, we verify the counterintuitive result that easy instances may have a much larger number of minimal forbidden sets. On the other hand, the trivial instances have a significanlty lower average number of unsolved minimal forbidden set than the non-trivial instances. However, it would be rash to conclude that the #UFS is a good indicator for measuiring instance hardess. Indeed, the number of unsolved minimal forbidden set remains relatively high for these trivial instances.

	$n+2$	$\|\mathcal{R}\|$	OS	NC	RF	RS	DR	#FS
Avg	62	4	0.39	1.78	0.66	0.67	0.39	826,181.59
Min	62	4	0.27	1.5	0.25	0.5	0.27	136
Max	62	4	0.53	2.11	1	0.72	0.54	12,630,775
σ	0	0	0.08	0.25	0.28	0.08	0.08	2,324,982.02

	ACFS	#UFS	ACUFS	OS^+	NC^+	DR^+	$\#FS^+$	$ACFS^+$
Avg	5.9	123.27	5.29	0.52	3.09	0.52	394,420.7	5.82
Min	3	6	2.55	0.35	2.18	0.35	55	2.93
Max	10.34	1733	10.34	0.7	4.19	0.7	10,338,140	10.38
σ	1.62	237.23	1.71	0.09	0.44	0.09	1,403,795.43	1.65

Table 7.8. *Indicators for 66 trivial KSD60 instances with unsolved minimal forbidden sets*

7.5.2. *Indicator stability and correlation*

In order to evaluate the stability of the indicators, we compare each indicator with its upper-bound based counterpart, by means of a correlation coefficient, average and standard deviation. We restrict this study to the KSD60 instance set. Since the number of minimal forbidden sets can be exponential in n, we take of the #FS and #FS+ indicators to perform the evaluations. The results are summarized in Table 7.9.

	NC/NC+	OS/OS+	DR/DR+	log#FS/log#FS+	ACFS/ACFS+
Correl. Coeff	0.25	0.75	0.75	0.98	1
Avg Dev (%)	52	23	23	6	1
Std Dev (%)	38	19	19	5	2

Table 7.9. *Impact of upper bounding on the KSD60 indicators*

The table shows that the NC indicator is the less stable indicator with important differences between instances representing the same problems and a weak correlation between NC and NC+. On the other hand, the upper-bounding procedure has no significant effect on the average cardinality of the minimal forbidden sets.

We illustrate the instability of NC in Figure 7.1. The X-axis represents the 480 KSD60 instances in the lexicographic order corresponding to increasing NC, then RF and then RS. The Y-axis gives the values of NC, NC+ and RS. One observes in the bottom part the variation of RS (0.2, 0.5, 0.7, 1.0) while the NC variation (1.5, 1.8, 2.1) is displayed in the middle part. The effect of the upper bounding procedure

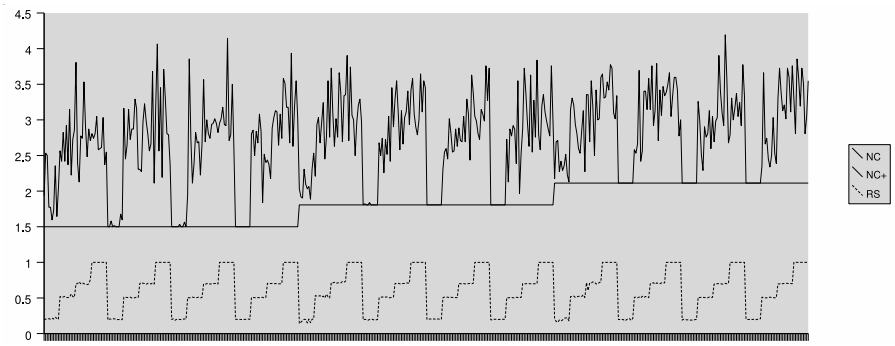


Figure 7.1. *The NC, NC+, and RS indicators for the 480 KSD60 instances*

is well illustrated by the observed relation between RS and NC+ (top part). When RS=0.2, the gap between NC and NC+ is very narrow whereas it becomes high for

the other values of RS. As already mentioned, several studies consider that the hardest instances are those with RS=0.2. Thus, the upper bound obtained by the simple heuristic greatly deviates from the critical-path lower bound which implies a smaller number of additional precedence constraints.

Figure 7.2 displays from bottom to top the values of RS, OS, OS+, and NC (RS being translated by -1 for reading convenience). The instances are sorted in the same lexicographic order. We observe that OS slowly increases with NC with small variations while OS+ also increases with NC and OS with more significant variations. Again, the deviation between OS and OS+ becomes zero when RS=0.2, which may lead to the same interpretation as for NC. DR and DR+ have the same stability as OS and OS+ since the disjunction ratio aggregates the precedence relations and the disjunctive relations, the latter being independent of the upper bound.

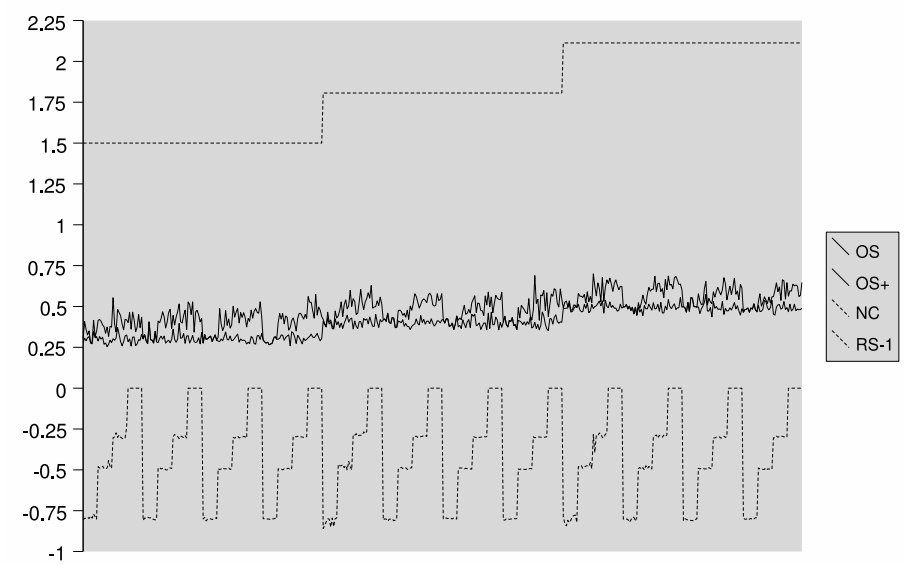


Figure 7.2. *The NC, OS, OS+, and RS indicators for the 480 KSD60 instances*

Focusing on the stability of the average number of minimal forbidden sets, the deviation between #FS and #FS+ also increases slightly as RS increases, with almost no difference for RS=0.2, as illustrated in Figure 7.3 where the instances are sorted according to RS. This is consistent with the results on NC, OS, and DR. However, the strong positive correlation between #FS and #FS+ is illustrated in Figure 7.4, where the instances are sorted in increasing Log#FS and where both Log#FS and Log#FS+ are displayed. Thus, the upper-bounding procedure for the minimal number of forbidden sets appears as a simplification for the computation of this indicator without questioning the discriminating power of both indicators.

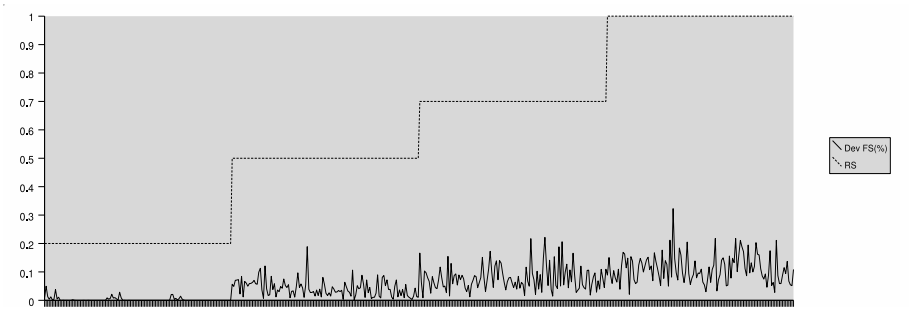


Figure 7.3. *The RS indicator and the deviation between #FS and #FS+*

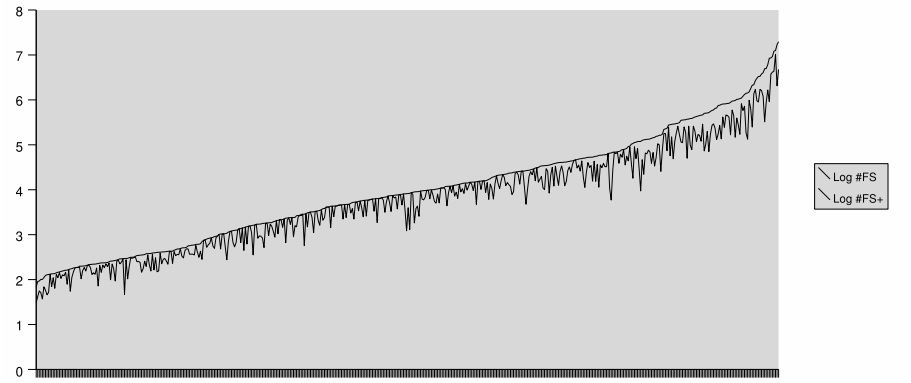


Figure 7.4. *Correlation between #FS and #FS+*

We now study the correlation between the minimal-forbidden-set based indicators (keeping only the upper-bound based ones) and the standard indicators NC, RS and RF. In Figures 7.5, 7.6 and 7.7, the values of RS, RF and NC have been translated for reader's convenience. Figure 7.5 displays, for all instances (sorted in the lexicographic order), the values of RS, RF, NC and Log#FS+ from bottom to top. As experienced by [STO 05] for #FS, #FS+ increases as RF increases and it increases as NC decreases while following a bell shape with respect to RS. Figure 7.6 replaces Log#FS+ by the average cardinality of minimal forbidden sets (ACFS+). Also, consistently with the results of [STO 05], ACFS+ increases as NC decreases and it increases as RS and RF increase.

Finally, Figure 7.7 shows how the number of unsolved minimal forbidden sets and their average cardinality evolve with the instance generation parameters. Globally, #UFS and ACUFS both decrease as NC increases while they both increase as RF increases. Moreover, #UFS tends to decrease as RS increases while ACUFS tends

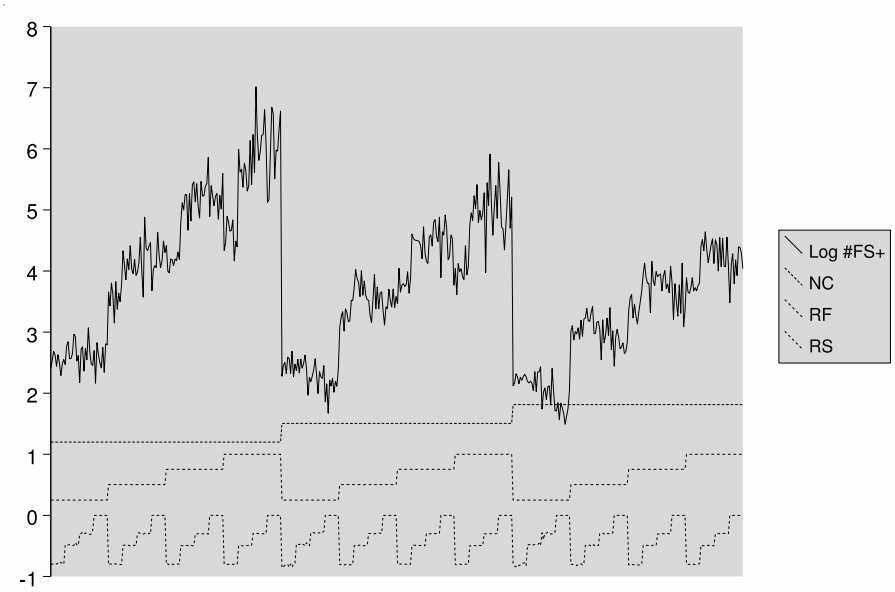


Figure 7.5. *The NC, RF, RS, and Log#FS+ indicators for the 480 KSD60 instances*

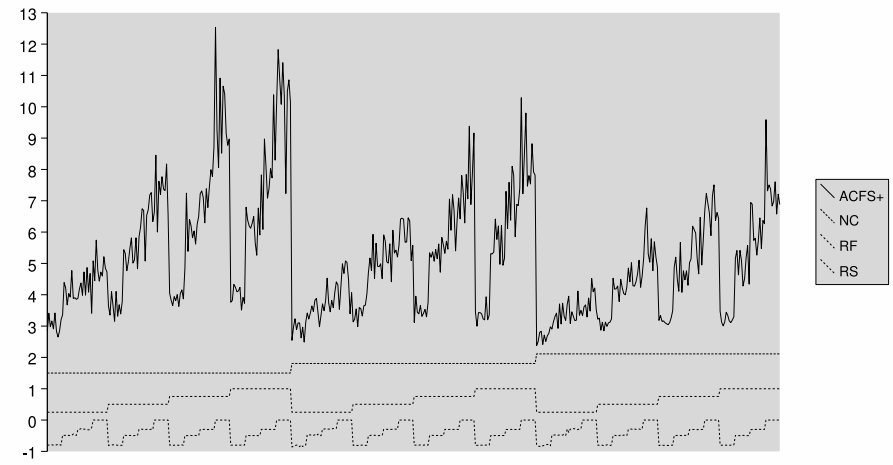


Figure 7.6. *The NC, RF, RS, and ACFS+ indicators for the 480 KSD60 instances*

to increase as the RS increases. Thus, we remark that #UFS is distinct from #FS for which the higher values correspond to medium RS. It could be that the #UFS is a good indicator of instance hardness (since again we known that the KSD60 instances with RS=0.2 are hard). However, this should be confirmed by computational experiments on solution methods.

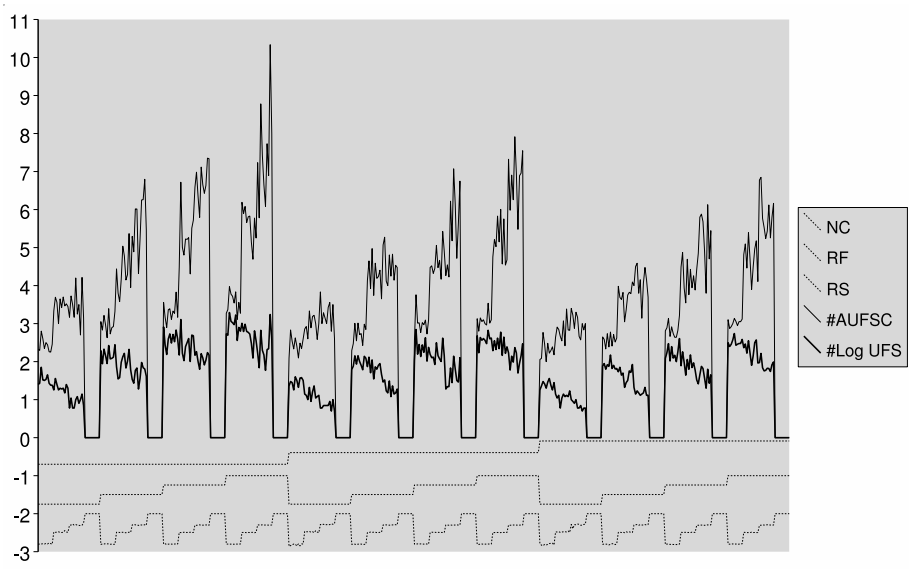


Figure 7.7. *The NC, RF, RS, #UFS and ACUFS indicators for the 480 KSD60 instances*

7.6. Comparison of solution methods

7.6.1. *Selected methods and experimental framework*

In this section, we present computational results on the KSD30, KSD60, BL and Pack instance sets from a selection of methods from the literature. The selected methods comprise lower-bound computation procedures, exact methods, heuristics and metaheuristics. We selected them because of their diversity and representativity of the different method categories presented in the preceding chapters.

LB0: *critical-path lower bound*

Among the lower-bound computation procedures, we select as a reference value the critical-path lower bound, denoted LB0, and defined as the length of the longest path in the precedence graph. The bound is very fast to compute: $O(|E|)$ with Bellman's longest-path algorithm. However, it can be expected to be far from the optimal makespan since it ignores resource constraints (see also section 2.1).

DCP: *disjunctive constraint-propagation based lower bound*

We choose a second lower-bound computation procedure based on constraint-propagation algorithms mostly focusing on disjunctive subproblem relaxations. We denote this bound DCP and we describe how it is computed below. As stated in section 2.2, the disjunctive subproblem relaxation consists of generating one-machine problems implied by the considered RCPSP instance. A one-machine problem consists of a set of pairwise disjunctive activities together with a time-window $[ES_i, LS_i]$ associated to each activity A_i, where ES_i (LS_i) denotes the earliest (latest) start time of A_i.

We set the time-windows as follows: for each activity A_i, the initial ES_i is set to the longest path from A_0 to A_i in the precedence graph, while LS_i is set to a certain upper bound minus the longest path from A_i to A_{n+1}. The upper bound, denoted UB, is obtained by the parallel scheduling algorithm with the MINLFT priority rule (see sections 6.2.1, 6.6.2 and 7.3).

The disjunction relation between two activities is established as follows: A_i and A_j are in disjunction if they cannot be processed in parallel. This occurs when there is a precedence relation ($A_i \prec A_j$ or $A_j \prec A_i$), when there is a resource $R_k \in \mathcal{R}$ such that $b_{ik} + b_{jk} > B_k$ or when the disjunction relation has been detected through the symmetric triple rule designed by Brucker et al. [BRU 98]. The latter rule is based on the following remark: suppose A_i, A_j, and A_k form a minimal forbidden set. Suppose that because of activity time-windows we know that (i) A_i and A_k must be processed in parallel from some non-zero time interval and (ii) A_j and A_k must be processed in parallel from some non-zero time interval. Then since $\{A_i, A_j, A_k\}$ is a minimal forbidden set, (i) and (ii) implies that there is a disjunction relation between A_i and A_j.

A one-machine problem corresponds to a clique in a graph where nodes correspond to activities and edges correspond to the disjunction relations. Unfortunately, there may be an exponential number of maximal cliques. To generate a limited number of one-machine problems, we use a randomized version of the heuristic proposed in [BRU 98]. The heuristic generates maximal cliques by sorting the activities in a random order and by generating greedily a maximal clique including the first activity, then for the next activity not included in a maximal clique, etc. For each different initial random order, we obtain a different set of maximal cliques (one-machine problems).

Then, for each one-machine problem, we apply the edge-finding constraint propagation technique to tighten the time-windows of the activity (see section 2.2). We also apply a lower bounding procedure by finding for each maximal clique $\mathcal{C}$ of activities

the largest bound given by

$$LB(\mathcal{C}) = \max_{\mathcal{C}' \subseteq \mathcal{C}} \left(\min_{A_i \in \mathcal{C}'} ES_i + \sum_{A_i \in \mathcal{C}'} p_i + \min_{A_i \in \mathcal{C}'} (UB - LC_i) \right), \qquad (7.1)$$

where LC_i is the latest completion time of A_i. Obviously, the magnitude of time-window adjustments and the lower-bound quality highly depend on the disjunctive nature of the considered instance, and also on the quality of the upper bound UB. We coded the DCP procedure in C++.

PWW: *linear programming lower bound [PRI 69]*

The third lower bound that we shall test corresponds to the continuous relaxation of the ILP proposed by Pritsker *et al.* [PRI 69] and described in section 3.2.1. The ILP comprises time indexed 0-1 variables representing the activities start times. We denote PWW the lower bound obtained by solving to optimality this LP relaxation. There is a pseudo-polynomial number of 0-1 variables since there is one binary variable per possible start time of each activity: at most $n(UB - p_i)$, where UB is the upper bound used as scheduling horizon. There is also a pseudo-polynomial number of resource constraints.

CAT: *linear programming lower bound [CHR 87]*

The fourth lower bound tht we shall evaluate is the continuous relaxation of the ILP proposed by Christofides *et al.* [CHR 87]. This ILP mainly proposes a disaggregated variant of the precedence constraints yielding a stronger relaxation at the expense of a pseudo-polynomial number of precedence constraints ($|E|UB$ in the worst case). We denote by CAT the lower bound obtained by solving to optimality this LP relaxation.

We coded all the linear programs using the C++ API of the ILOG Cplex 10.2 linear programming solver [ILO07b]. We used the default Cplex parameters for linear programming.

DCP-PWW and DCP-CAT: *constraint propagation/linear programming lower bound*

The size of the PWW and CAT linear programs strongly depend on the number of possible activity start times. Thus, it seems natural to use the time-window reductions performed by constraint propagation as a preprocessing step of the LPs. This reduces the number of 0-1 variables to $LS_i - ES_i$ for each activity, and also the number of precedence constraints in the disaggregated model. This also restricts the feasible set of continuous solutions, which in turn possibly increases the optimal relaxed makespan. Such preprocessing techniques and other CP-based valid inequalities have been studied in [DEM 02a, DEM 05] and also reported in section 3.2.1. We denote by

DCP-PWW and DCP-CAT the solving of the Pritsker *et al.* [PRI 69] and Christofides *et al.* [CHR 87] LP relaxation after DCP preprocessing.

MCS: *complete minimal critical set lower bound [LAB 05]*

The last lower bound computation method that we shall test is also an exact method: the complete MCS-based search proposed by Laborie [LAB 05], where MCS stands for Minimal Critical Set, another designation of the minimal forbidden set concept. Following the destructive framework, the method applies a complete backtrack search with constraint propagation to verify whether there exists a solution satisfying a tentative upper bound T, where T is initially set to a valid lower bound. If the search fails then $T + 1$ is a lower bound and the process can be iterated by setting $T \leftarrow T + 1$. If the search succeeds, T is the optimal makespan. The backtrack search proceeds by detection and resolution of minimal forbidden sets. A resolver of a minimal forbidden set is a precedence relation that breaks up the forbidden set. Namely, each node maintains a set of precedence relations between the activities. During node evaluation, one selects a minimal forbidden set violated by the precedence constraints. Children nodes are created by generating for each of them a precedence relation (a resolver) breaking up the forbidden set. The minimal forbidden set is selected so as to minimize an estimation of the remaining search space after its resolution. The method also comprises a self-adapting shaving technique applied at each node to strengthen constraint propagation. If all but one resolvers of a minimal forbidden set yield a search failure after constraint propagation, then the precedence constraint of the remaining resolver must be verified. The constraint propagation techniques used are the timetable, disjunctive, edge-finding, precedence-energy and balance constraints (described in Chapter 4). A time limit is imposed to the backtrack search so that the current lower bound is returned if the time limit is reached. If the time limit is not reached, the optimal makespan is returned. The method, based on the ILOG Constraint Programming solver was kindly provided by Philippe Laborie, as a binary executable file using ILOG CP 1.2 [ILO07a] runtime license.

DH: *branch-and-bound [DEM 97]*

Our computational experiments also comprise the exact branch-and-bound method proposed by Demeulemeester and Herroelen [DEM 97], denoted by DH. The method is based on the delaying alternative branching scheme (see section 5.1.3.1) and uses the local left shift dominance rule (see section 5.1.2), the powerful cut-set dominance rule (see section 5.1.4), and the weighted node packing bound inspired by Mingozzi *et al.* [MIN 98] (see section 3.2.2). As stated in section 7.4, this method was the first to solve all KSD30 instances to optimality. The method was kindly provided by Erik Demeulemeester and Willy Herroelen as C++ files.

PAR-LFT: *parallel schedule scheme with the MINLFT priority rule*

Besides lower-bound computation and exact methods, we shall also evaluate three heuristics and metaheuristics. The first one is a simple call of the parallel scheduling scheme with the minimum latest finishing time priority rule. We used this method that is described in detail in sections 6.2.1 and 6.6.2, to compute the upper-bound based indicators of section 7.3 and the time-windows for the disjunctive relaxations above. Since it is very fast ($O(n^2m)$ time complexity), it is not expected to obtain the best results but rather serves as a reference method. It is referred to as PAR-LFT. We coded it in C++.

TABU and TABU+: *tabu seach [PIN 94]*

The second heuristic we shall use is the tabu search method of Pinson *et al.* [PIN 94], described in section 6.4.2 and based on the activity-list representation and on pairwise interchange, swap and shift neighborhood (see section 6.3.1). One generates a schedule from the list by means of the serial scheduling algorithm (see sections 6.2.1 and 6.6.1). Being designed in 1994, this method does not incorporate the up-to-date improvements described in section 6.3.1. However, it is noteworthy that this kind of procedure is still the backbone of many of the most successful heuristics at the moment and the method remains representative of the activity-list based metaheuristics. The method can be stopped after a maximum number of calls of the serial SGS or after a time limit. We denote by TABU the Pinson *et al.* heuristic stopped after 50,000 calls to the SGS using a tabu list size of 50 moves. We denote by TABU+ a version of the method with the same tabu list size and a maximum allocated CPU time equal to the maximum CPU time required by the LNS method described below, for sake of comparison. The method was kindly provided by Eric Pinson and David Rivreau as C++ files.

LNS: *large neighborhood search [PAL 04]*

The last heuristic that we shall evaluate is the large neighborhood search method of Palpant *et al.* [PAL 04], also described in section 6.4.4. It can be defined as a multistart randomized large neighborhood search method using the block-based neighborhood (see section 6.3.4). At each iteration, a subproblem is obtained from the current complete schedule by freezing a subset of activities to their current start time value. The other activities form a block of activities which define a potentially exponential-sized neighborhood and is entirely rescheduled by means of constraint programming techniques. There is an arbitrary limit set on the total number of iterations. The block of activities is generated randomly. We selected this heuristic because is not based on the activity list representation. We denote this method by LNS. We use the C++ code developed by Palpant *et al.* [PAL 04] using ILOG CP 1.2 to reschedule the successive activity blocks.

Experimental framework

In the following section, we detail and comment the results of the above-selected methods on each of the selected instance set (KSD30, KSD60, BL and Pack). We carried out all computational experiments on a Dell PC equipped with a XEON 5110 biprocessor clocked at 1.6Ghz with 2GB RAM.

When the optimal makespan is known (for the BL and KSD30 instances) we report the deviation ΔOPT as a percentage of the obtained lower or upper (depending on the method category) bound from the optimal makespan , as well as the deviation $\Delta LB0$ above the critical-path lower bound LB0, in terms of average, minimum, maximum and standard-deviation values. We also report the CPU time in secondq, the percentage #opt of optima found, the percentage #best of best lower (or upper) bound found, and, for the two exact methods (MCS and DH), the number #proved of optimum found and proved.

7.6.2. *Results on the KSD30 instances*

Table 7.10 reports the comparative results of the different evaluated methods on the KSD30 instances.

Comparing exact methods, the DH methods solves all instances to optimality with a very low average computational requirement (0.07 s) with a maximum CPU time of 8.65 s. The MCS method (with a time limit of 1,000 s) solves 469 instances out of 480. The instances unsolved by MCS all belong to the category RS=0.2 and RF=1 (except one for which RF=0.75) with a number of unsolved minimal forbidden set ranging from 48 to 159 (the average number of unsolved minimal forbidden sets is 36 for the KSD30 instances with RS=0.2).

The following ranking can be observed for the lower bounds obtained in terms of relaxation quality: MCS > DCP-CAT > DCP-PWW > DCP > CAT > PWW > LB0.

For the CPU time requirements, the ranking becomes LB0 < DCP < DCP-PWW < PWW < DCP-CAT < CAT < MCS. It follows that CAT is dominated by DCP-CAT and PWW is dominated by DCP-PWW. Indeed, the preprocessing based on constraint propagation drastically reduces the CPU time for solving the linear programming relaxations while the lower bound is increased by more than 100%. Therefore, in the remaining of the chapter, we display only the results of DCP-CAT and DCP-PWW neglecting CAT and PWW. Note that MCS could be launched with much less allowed CPU time, which could modify the ranking especially when compared with the CAT bound. There is also an important gap between the best bound (provided by MCS, which is also an exact method) and the second best bound (provided by DCP-CAT).

The TABU and LNS heuristic obtain a deviation from the optimal solution of less than 0.4%. This especially underlines the quality of the TABU search heuristic which

obtains an average deviation from the optimum of 0.39 with an average CPU time of 0.2 seconds. The ranking for the makespan quality is LNS < TABU < TABU+ while for the CPU time the ranking becomes TABU < LNS < TABU+.

		LB0	DCP	PWW	DCP-PWW	CAT	DCP-CAT	MCS	DH	PAR-LFT	TABU	TABU+	LNS
ΔOPT	Avg	9.21	3.86	8.74	3.81	7.37	3.48	0.08	0.00	4.53	0.39	0.34	0.06
(%)	Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	54.72	33.70	47.83	33.70	40.22	28.26	8.00	0.00	32.81	8.51	8.51	3.45
	Stdev	13.32	6.43	12.27	6.31	10.38	5.69	0.62	0.00	5.95	1.25	1.17	0.33
$\Delta LB0$	Avg	0.00	7.94	0.81	8.01	2.78	8.45	13.23	13.37	19.31	13.92	13.85	13.47
(%)	Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	0.00	102.08	28.13	102.08	37.50	102.08	114.58	120.83	143.75	131.25	131.25	120.83
	Stdev	0.00	15.73	3.30	15.72	5.83	15.93	22.00	22.43	29.36	23.32	23.19	22.69
CPU	Avg	0.00	0.09	0.52	0.07	4.88	0.67	10.62	0.07	0.00	0.24	17.52	4.21
(s)	Min	0.00	0.07	0.02	0.02	0.10	0.05	0.02	0.00	0.00	0.00	0.00	0.00
	Max	0.00	0.12	8.95	3.45	205.66	23.04	1002.16	8.65	0.00	0.85	71.00	65.00
	Stdev	0.00	0.01	1.34	0.17	14.97	2.16	57.55	0.46	0.00	0.34	30.39	11.79
#opt (%)		45.00	58.96	45.00	58.96	46.04	59.79	97.92	100.00	44.38	87.71	88.75	96.67
#best (%)		45.00	58.96	45.00	58.96	46.04	59.79	97.92	100.00	44.38	87.71	88.75	96.67
#proved (%)		-	-	-	-	-	-	97.71	100.00	-	-	-	-

Table 7.10. *Method comparison on the 480 KSD30 instances*

The average results have to be tempered by the fact that, as already emphasized, a large proportion of the KSD30 instances are trivial. Let us consider now the 120 instances having RS=0.2 for the four possible values of RF. Figure 7.8 shows the average deviation from the optimum of all methods. One observes the ranking of the methods remains the same. However, the gaps of the lower bounds are much larger than the average value over the 480 instances. For example, the second best lower bound is almost at 15% from the optimum for the instances with RF=1. The gap also increases as RF increases. These results are consistent with previous studies. We also observe the quality of the DCP bound, mostly based on disjunctive reasoning on the KSD instances with RS=0.2. For these instances, the improvement brought by the LP-relaxation is rather weak, especially for instances with RF < 1 and the PWW relaxation. The tabu search performance remains satisfactory with about 1% deviation from the optimum while the LNS optimality gap increases to 0.23%.

To further study the influence of RF and other indicators with RS=0.2, we report in Table 7.11, the CPU times of the methods along with the numbers (#FS+) and average cardinalities (ACFS+) of upper bound based minimal forbidden sets, the number (#UFS) and average cardinality (#ACUFS) of unsolved minimal forbidden sets, in terms of average values in function of RF. We observe that for methods PWW, DCP-PWW, CAT, DCP-CAT, MCS, DH and LNS, the CPU time significantly increases when RF increases. On the other hand, for all methods except DH the solution quality decreases. We also observe that all the averaged minimal forbidden set indicators also increase in terms of number and average cardinality.

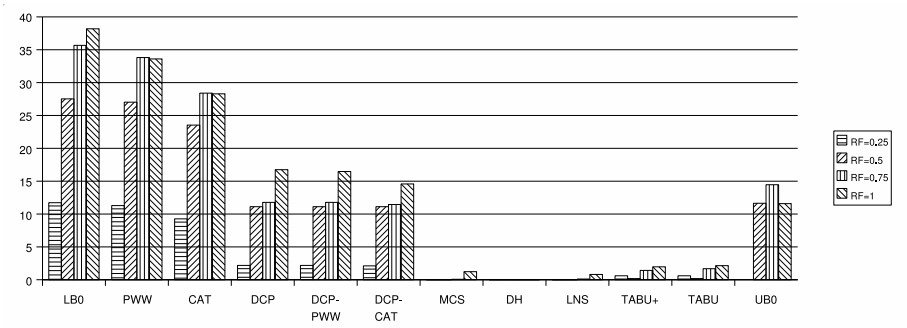


Figure 7.8. *Average optimality gaps on the KSD30 instances (RS=0.2) depending on RF*

RF	#FS+	ACFS+	#UFS	AC-UFS	LB0	DCP	PWW	DCP-PWW	CAT	DCP-CAT	MCS	DH	UB0	TABU	TA-BU+	LNS
0.25	33	2.22	10	2.14	0.00	0.09	0.04	0.03	0.70	0.15	0.05	0.00	0.00	0.01	0.01	0.00
0.5	136	2.54	26	2.24	0.00	0.09	0.40	0.07	7.36	0.45	4.87	0.02	0.00	0.77	68.62	2.13
0.75	238	2.59	43	2.35	0.00	0.09	2.67	0.13	20.30	1.71	79.46	0.18	0.00	0.78	70.94	22.97
1	380	2.66	66	2.48	0.00	0.09	3.98	0.35	36.87	6.61	116.28	0.76	0.00	0.77	70.93	40.13

Table 7.11. *CPU times and MFS indicators on the KSD30 instances (RS=0.2) depending on RF*

7.6.3. *Results on the BL instances*

Table 7.12 displays the method results on the BL instances. Both the DH and MCS exact methods solve all instances to optimality. This is partly linked to the relatively small size of the instances (20 or 25 activities). The CPU time needed by DH is negligible. Contrarily to the KSD30 instance set, we could not find any correlation between the CPU time required by the MCS method and the indicators, although the instances can be distinctly classified in two groups : 6 instances need more than 10 seconds while the remaining instances are solved in less than 2.5 seconds. If we look at the distribution of the indicators, we also clearly have a smaller variability than for the KSD30 set: RS varies from 0.16 to 0.55, while RF ranges between 0.5 and 0.77. The average number of minimal forbidden sets is also larger than that of the hardest KSD instances (RS=0.2, RF=0.1) (see Table 7.3). Thus, we would need other indicators to explain the difficulty of these instances.

For the lower bound, the DCP bound dramatically decreases: it is equal to LB0. This is due to the absence of relevant one-machine relaxations in the BL instances. As a counterpart, the complementarity of the LP relaxations is emphasized. The CAT bound is twice as close to the optimum than the LB0 bound. Note that other

constraint-programming based methods could be used as more efficient preprocessing techniques. To mention one, Baptiste and Le Pape [BAP 00] demonstrated the efficiency of energetic reasoning techniques on theses instances.

For the heuristics, the tabu search method obtains better results than for the hard KSD 30 instances. On the other hand, the results of the LNS method are twice worse on average than for the KSD instances with RS=0.2. The subproblems solved at each iteration of the LNS method seem harder than those solved for the KSD30 instances.

		LB0	DCP	DCP-PWW	DCP-CAT	MCS	DH	PAR-LFT	TABU	TABU+	LNS
ΔOPT (%)	Avg	22.62	22.62	17.41	11.17	0.00	0.00	9.11	0.71	0.62	0.52
	Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	46.67	46.67	33.33	26.67	0.00	0.00	27.78	5.26	5.26	5.26
	Stdev	11.95	11.95	7.94	6.54	0.00	0.00	6.15	1.56	1.49	1.40
$\Delta LB0$ (%)	Avg	0.00	0.00	7.92	16.41	32.40	32.40	44.25	33.41	33.30	33.12
	Min	0.00	0.00	0.00	0.00	0.00	0.00	13.79	0.00	0.00	0.00
	Max	0.00	0.00	33.33	40.00	87.50	87.50	100.00	87.50	87.50	87.50
	Stdev	0.00	0.00	9.62	11.60	21.33	21.33	23.47	22.00	22.07	21.72
CPU (s)	Avg	0.00	0.04	0.02	0.07	3.29	0.01	0.00	0.46	30.96	7.18
	Min	0.00	0.02	0.00	0.01	0.03	0.00	0.00	0.37	30.19	0.00
	Max	0.00	0.06	0.05	0.28	35.71	0.08	0.00	0.58	31.00	32.00
	Stdev	0.00	0.01	0.01	0.06	7.44	0.02	0.00	0.07	0.13	11.17
#opt (%)		2.56	2.56	2.56	5.13	100.00	100.00	10.26	82.05	84.62	87.18
#best (%)		2.56	2.56	2.56	5.13	100.00	100.00	10.26	82.05	84.62	87.18
#proved (%)		-	-	-	-	100.00	100.00	-	-	-	-

Table 7.12. *Method comparison on the 39 BL instances*

7.6.4. *Results on the KSD60 instances*

Table 7.13 displays the results of the methods on the KSD60 instances.

For the exact methods, while the MCS method was clearly outperformed by the DH method for the KSD30 and BL instances, it obtains in turn better results on this instance set. Although MCS requires more CPU time than DH in average to solve the instances to optimality (31 s vs. 13 s), it solves more instances (402 vs 384) within a time limit of 3,000 s for both methods. The compared results of MCS and DH presented so far show the interest of developing new hybrid exact methods to solve the RCPSP. Combining the power of constraint propagation and self-adapting shaving with efficient lower bounds and powerful dominance rules could be a way of improving these results.

For the lower bounds, the results are consistent with those for the KSD30 set, although the CPU times of the linear programming relaxation become high. For the DCP-CAT lower bound, the CPU time can reach 916 seconds; it is 25.8 seconds on average. For the DCP-PWW lower bound the CPU time remains reasonable but is

still high (up to 64 s and 1.85 s on average). This obviously questions the relevance of integer linear programming branch-and-bound for solving the RCPSP. However, the reasonable quality of the best LP-based bound (combined with the DCP preprocessing) together with the recent success of column-generation-based [BRU 00, BAP 04] and lagrangean relaxation-based lower bounds [MÖH 03] makes room for further improvements, and again hybrid methods. Thus, combining constraint propagation with branch-and-price and/or branch-and-cut techniques appears to be another promising way of tackling the RCPSP.

For the heuristics, the best results are obtained according to the following rank with respect to the average gap ΔUB^* with the best upper bound LNS > DH > TABU+ > TABU > PAR-LFT. We have to stress the excellent result of the DH method as a heuristic. However, one must here take into account the high average CPU time (600 s) obtained by incorporating instances for which the 3,000 s limit allocated to the DH method is reached (as for the exact methods, the tables display the average CPU time for solved instances only). The TABU method definitely offers a good compromise between the quality of the solution (about 1% above the best upper bound on average, with a maximum of 12%) and the CPU time: less than one second. The TABU+ method significantly improves the results of the TABU method with a rather high average CPU time (35 s) without reaching the result quality of the LNS method. The LNS method is particularly successful on the KSD60 instance set: it obtains the best result in 94% of the cases and it is always below 2.6% from the best upper bound. However, LNS has a rather high average CPU time (13 s) and we have to emphasize that state-of-the-art metaheuristics (see Chapter 6 and [KOL 06]) are able to reach comparable results with much less required CPU time. Nevertheless, the table gives a good idea of today's maximal gap between the best known lower bound and the best known upper bound on the KSD60 instances which reaches 22%.

Let us now use the fact that several instances are still unsolved in order to evaluate further the relevance of the different indicators with respect to instance difficulty. First, we remove all KSD60 easy instances: those solved by the DH procedure in less than 1 second. Then, we compute for the DH procedure the average indicator values for the unsolved and solved non-easy instances. Table 7.14 displays the results: columns S refer to the solved instances (with at least 1 s CPU time for DH) while columns U refer to the unsolved instances. Columns marked with UB refer to the upper-bound based variant (+). The last row displays the gap between the initial trivial lower and upper bound LB0-UB0/LB0, which we shall refer to in the sequel as the initial duality gap. The table shows that unsolved instances tend to have lower OS, NC, DR and RS, and higher RF, #FS, ACFS, #UFS and ACUFS than the unsolved instances. This confirms the result obtained for the KSD30 instances. However, the most discriminating indicator appears to be simply the initial duality gap, which can be easily computed and which is also rather intuitive!

		LB0	DCP	DCP-PWW	DCP-CAT	MCS	DH	PAR-LFT	TABU	TABU+	LNS
ΔUB^* (%)	Avg	7.21	5.47	4.98	4.22	1.39	0.52	4.96	1.12	0.65	0.07
	Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	51.27	47.50	36.05	29.93	17.46	10.31	26.19	12.50	10.45	2.56
	Stdev	13.18	11.02	9.62	7.91	3.79	1.53	5.85	2.07	1.52	0.30
ΔLB^* (%)	Avg	6.22	4.42	3.86	3.01	0.02	2.13	6.70	2.74	2.26	1.63
	Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	45.00	42.73	28.00	21.82	2.56	26.02	42.86	27.84	25.96	22.12
	Stdev	11.21	8.66	7.13	5.48	0.19	5.52	9.09	5.88	5.33	4.39
$\Delta LB0$ (%)	Avg	0.00	2.47	3.31	4.51	8.59	11.60	17.12	12.37	11.75	10.91
	Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	0.00	56.96	56.96	56.96	81.82	112.99	136.36	115.58	112.99	105.19
	Stdev	0.00	6.42	7.36	9.56	16.71	23.38	28.66	24.17	23.35	21.83
CPU (s)	Avg	0.00	0.52	1.45	25.80	31.14	12.89	0.00	0.95	35.33	13.86
	Min	0.00	0.46	0.06	0.04	0.04	0.00	0.00	0.00	0.00	0.00
	Max	0.00	0.64	63.85	916.79	2904.11	2025.37	0.00	2.20	101.00	86.00
	Stdev	0.00	0.02	5.52	93.12	168.92	139.23	0.00	0.89	46.14	24.23
#opt (%)			66.46	66.46	66.67	84.58	81.67	40.63	69.58	74.58	81.88
#best (%)			66.88	66.88	67.29	99.17	84.79	40.63	70.00	77.08	94.79
#proved (%)		-	-	-	-	83.75	80.00	-	-		-

Table 7.13. *Method comparison on the 480 KSD60 instances*

			UB(+)	
	S	U	S	U
OS	0.42	0.38	0.45	0.40
NC	1.87	1.76	2.29	1.94
DR	0.46	0.43	0.49	0.44
RF	0.56	0.81	-	-
RS	0.34	0.26	-	-
Log#FS	3.47	4.07	3.40	4.04
ACFS	3.89	3.95	3.87	3.94
Log #UFS	1.84	2.44	-	-
ACUFS	3.26	3.48	-	-
(UB0-LB0)/LB0	27%	62%	-	-

Table 7.14. *Comparison of average indicator values between solved and unsolved KSD60 instances*

7.6.5. *Results on the Pack instances*

Table 7.15 displays the results of the methods on the Pack instances. The instance parameters did not allow to apply directely the available version of the DH procedure code. The MCS method is able to solve only 19 instances out of 55 within a time limit of 3,600 seconds. We remark that these instances are characterized by a small RS, a high RF, a small OS and a large number of (total and unsolved) minimal forbidden sets of average cardinality of more than 3.1 activities. The initial duality gap if of 215%. Although the number of instances is relatively small, most of the indicators signal hardness. However, we are not able to distinguish between hard and easy instances from this category through the indicator values. The apparently pertinent initial duality

gap indicator 207% for the solved instances and 221% for the unsolved instances. This again emphasizes the need for more trustable hardness indicators.

		LB0	DCP	DCP-PWW	DCP-CAT	MCS	PAR-LFT	TABU	TABU+	LNS
ΔUB^* (%)	Avg	60.53	26.42	20.98	17.44	3.84	6.60	0.38	0.21	0.39
	Min	11.54	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	83.33	70.31	49.28	46.88	20.83	16.67	4.60	1.96	4.55
	Stdev	16.60	23.80	16.95	15.01	6.17	4.51	4.60	1.96	4.55
ΔLB^* (%)	Avg	58.92	24.57	18.73	15.03	0.63	10.40	4.10	3.92	4.11
	Min	5.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Max	83.33	65.52	41.38	37.93	20.83	26.44	25.45	25.45	25.45
	Stdev	17.57	22.61	15.09	12.90	3.08	6.02	25.45	25.45	25.45
$\Delta LB0$ (%)	Avg	0.00	128.94	142.99	151.62	184.61	215.92	197.02	196.47	196.90
	Min	0.00	0.00	0.00	0.00	5.80	15.94	13.04	13.04	13.04
	Max	0.00	500.00	500.00	500.00	425.00	500.00	500.00	500.00	500.00
	Stdev	0.00	141.18	130.16	127.57	110.96	123.06	114.45	114.05	114.18
CPU (s)	Avg	0.00	0.05	0.21	1.90	493.50	0.00	0.55	59.86	24.96
	Min	0.00	0.01	0.01	0.05	0.04	0.00	0.06	0.06	1.00
	Max	0.00	0.12	1.42	18.00	3020.72	0.00	0.89	60.99	61.00
	Stdev	0.00	0.03	0.36	3.17	777.32	0.00	0.16	8.21	12.87
#opt (%)		0.00	10.91	10.91	12.73	52.73	1.82	47.27	49.09	50.91
#best (%)		0.00	12.73	12.73	16.36	92.73	5.45	83.64	87.27	85.45
#proved (%)		-	-	-	-	34.55		-		-

Table 7.15. *Method comparison on the 55 Pack instances*

7.7. Conclusion

We tested a representative variety of lower-bound computations, exact methods and heuristics on several instance sets from the literature. First, we compared the performance of these methods. Secondly, we evaluated the ability of several indicators to characterize instance hardness. On the KSD instance set, we were able to confirm previously-established as well as new correlations that can also explain the hardness of the Pack instances. However, our conclusion is that generalizing these results is somewhat rash and there is still a need for new propositions of indicators of instance hardness. This is corroborated by the counterexamples that we highlighted both with easy and hard instances, and by our evaluation of the stability of the various indicators. Finally, a reasonable hardness indicator appears to be simply the initial duality gap, which has the not insignificant advantage of being quickly computable. However, our computational results, especially those on the KSD60 and Pack instance duality gaps, illustrate the limits of current state-of-the-art RCPSP solution methods, and provide guidelines for future research.

SECOND PART

Variants and Extensions

Chapter 8

Preemptive activities

8.1. Preemption in Scheduling

This chapter describes a classical extension of scheduling problems: the fact that the execution of an activity may be interrupted by the execution of an other one, and continued later. Such an activity is called *preemptive*, and we assume here that no penalty is given if any interruption occurs. A project in which all activities are preemptive is also called preemptive. Some contributions have treated derived cases of preemption: either interruption can occur only in favor of specific activities (*selective preemption*), or an activity can be interrupted most once or twice (*limited preemption*), or, in more recent works, only a subset of activities are preemptive. Allowing preemption improves evidently the optimal objective value of a scheduling problem, as some constraints are relaxed. But it also increases a lot the cardinality of the feasible solution set.

In the following, we distinguish two specific kinds of preemption. Indeed, the very few approaches dealing with preemption for the RCPSP consider all integer dates of the project to interrupt the execution of some activities. This is called *integral preemption* and this can be done since all processing times are integer or can tolerate a common discretization. To an industrial point of view, it seems to be convenient to schedule parts of activities for a given number of time units (CPU clock for microprocessor instructions, days or half-days of work on a project ...). To a theoretical point of view, Baptiste et al. [BAP 05] claim that a wide class of scheduling problems admits, for a lot of objective functions, an optimal preemptive solution such that all interruptions occur at integral dates. This is not the case of makespan minimization,

Chapter written by Jean DAMAY.

which will be considered hereafter. For example, consider the instance containing one resource with a capacity of two units and three *incomparable* activities (with no precedence relations between them) of duration one time unit, and of requirement one resource unit ($q = 1, B_1 = 2, n = 3, E = \emptyset$, and $\forall A_i \in \mathcal{A}, b_{i1} = 1, p_i = 1$) All optimal solutions of this instance have their first interruption at time 0.5. Hence, an alternative approach consists in considering *rational preemption*,for which all preemptive activities can be interrupted at any rational times during the project. The resolution of such problems involves mathematical programming techniques, as it seems difficult to predict iteratively at what time the next preemption could occur.

Note that the exact resolution of any kind of preemptive version mentioned above provides a lower bound of the optimal makespan of the non-preemptive one. Some major contributions [MIN 98] [BRU 00] [BAP 04] obtain high-quality lower bounds of such values from a formulation that relaxes these non-preemption constraints, but also partially the precedence ones (see section 3.2.2). To our knowledge, no work had been done to solve the rational preemptive version of the RCPSP.

8.2. State of the art

8.2.1. *Integral preemption for RCPSP*

In several approaches dealing with the non-preemptive case, the algorithm consists in adding iteratively activities or set of activities to partial schedules (cf list algorithms or chronological branching schemes of section 5.1), in order to build a complete schedule. If this algorithmic scheme is applied for the preemptive case, it is not sufficient only to consider the finishing times of each activity to enrich the partial schedule.

Le Pape and Baptiste present in [Le 98] a few constraint propagation techniques to the preemptive version of some disjunctive scheduling problems. For cumulative ones (such as the RCPSP), one can clearly see that the propagation process would be rather slow, since prevents from using some simple propagation techniques.

We will focus here on the main contribution for the integral preemptive RCPSP: the paper of Demeulemeester and Herroelen [DEM 96a]. Derived from their approach for the non-preemptive version [DEM 92b] described in section 5.1.1.2, the method they implement is a branch-and-bound procedure in which each node corresponds to a partial feasible schedule composed of parts of activities scheduled before a date (or *decision point*) $t \in \mathbf{N}$. In the leaves of this tree, each activity is entirely processed, so they correspond to complete schedules.

The branching scheme consists in extending a partial schedule, hence finding all subsets of the so-called *eligible* activities that could be in process on the next time interval and lead to the optimal solution derived from this node. Indeed, the decision

point of the child nodes is $t+1$. Hence, it does not only correspond to the completion date of the activities, like Kaplan suggested it in [KAP 88]. Indeed, the authors split each activity A_i in p_i unit duration activities and apply almost the same algorithm of [DEM 92b] for the non-preemptive version. Thus, they can adapt and provide some dominance rules to restrict the calculation of the next feasible subsets: because of resource constraints, one can first identify activities that can be in process on the next interval(s) and lead to an optimal solution, and secondly determine all minimal (in the sense of inclusion) subsets of activities the execution of which can be delayed. They also propose a simple comparison with previously saved nodes to eliminate some of them, and local lower bound processes: the most intuitive lower bound calculation corresponds to t augmented by the length of the remaining critical path, and it is the only one tested in practice.

The computational results of this algorithm are presented for the Patterson set of 110 instances, which are all optimally solved within 5 minutes. Preliminary numerical results of a recent reimplementation with new dominance rules may indicate that this algorithm requires too many CPU-time to provide the optimal solutions of a lot of PSPLIB instances with 30 activities. Note that the Patterson instances are known to belong to the set of the "very easy"ones (cf [KOL 95b]).

8.2.2. *Rational preemption for parallel machine scheduling problems*

Note first that research about preemption in scheduling problems mainly concerns the machine scheduling problems (job-shop, flow-shop , parallel machine ...). A classical result comes from McNaughton [MCN 59] for the parallel machine scheduling problem and provides an algorithm to build an optimal schedule with at most $m-1$ interruptions when m incomparable activities have to be executed (such as in the example mentioned in 8.1).

Derived from Papadimitriou and Yannakakis [PAP 79], we provide here the definition of the *interval orders* in scheduling problems. We say that a precedence graph $G(V,E,p)$ belongs to the interval order class if each vertex of V, that is to say each activity, can be associated to an interval in the real line such that $(A_i,A_j) \in E$ iff every interior point of the associated interval of A_i is strictly lower than every interior point of the one associated to A_j. In practice, these intervals correspond to the smallest closed time windows $[ES_i;LC_i], A_i \in V$. The authors provide a polynomial (linear) algorithm to solve the parallel machine instances with such a precedence graph and unit-time tasks.

Sauer and Stone [SAU 89b] [SAU 89a] extend this polynomiality result to the preemptive case of this problem. In [MOU 99], Moukrim introduce a new class of orders, called *quasi-interval orders*, the instances of which are solved by the algorithm described by Papadimitriou and Yannakakis in its preemptive version, and still with unit-time tasks.

But the authors of [SAU 89b] use a rather new linear programming formulation, in which the variables are associated to Φ the set of all the so-called *feasible subsets* of activities (see section 3.2.2), i.e. the subsets of activities that can be simultaneously in process during a feasible schedule ($a_l \in \Phi$ iff $\forall R_k \in \mathcal{R}, \sum_{A_i \in l} b_{ik} \leq B_k$ and $\forall A_i, A_j \in l, (A_i, A_j) \notin TE$ and $(A_j, A_i) \notin TE$, where TE is the transitive closure of E). The program, called $\mathcal{P}$ in this chapter, is presented in the next section as an adaptation of a relaxation of Mingozzi et al. [MIN 98] for the RCPSP (with arbitrary processing times). Sauer and Stone also define a precedence relation $\prec$ on Φ: $\forall a_l, a_{l'} \in \Phi, a_l \prec a_{l'}$ iff $\exists A_i \in a_l, A_j \in a_{l'}$ such that $(A_i, A_j) \in E$) and identify that the lack of cycles in the graph $\Gamma = (\Phi, \prec)$ is a crucial feasibility property. Since the interval orders verify this property, the solution of the linear program $\mathcal{P}$ provides the optimal preemptive solution of the instances having such precedence graphs.

In [MOU 97] [MOU 05], Moukrim and Quilliot identify an other class, called $\mathcal{A}_m$*-order*, of parallel machine instances for which the solution of $\mathcal{P}$ is equal to the makespan of the optimal preemptive solution. They provide in [MOU 05] an algorithm to build such an optimal preemptive schedule from this solution of $\mathcal{P}$, for this class. In [MOU 97], the authors reuse this linear formulation to propose a descent algorithm based on the Simplex pivoting for parallel machine problems (for both preemptive and non preemptive case). Note that they reuse works on hypergraphs from Bendali and Quilliot [BEN 97]. All this is reconsidered in the next section for the RCPSP.

8.3. Recent LP-based methods

In the following, our recent works dealing with the preemptive version of the RCPSP [DAM 08] [DAM 05] are summarized. Indeed, the following results are initially provided for a given subset of preemptive activities, but we assume here that they all are preemptive, for sake of simplicity. The descent approach is derived from the works of [MOU 97] mentioned in the previous section dedicated to parallel machine scheduling problems. Note that all propositions and theorems presented below are given without proof, but are intuitively assumable.

8.3.1. *Reformulation*

Let us introduce some notations dealing with the set Φ of feasible subsets of activities, that we call *valid antichains* (the corresponding subset of vertices are antichains of the precedence graph): $\forall a_l, a_{l'} \in \Phi, a_l \prec a_{l'}$ iff $\exists A_i \in a_l, A_j \in a_{l'}$ such that $(A_i, A_j) \in E$. Note that $\prec$ is not an order relation, and we call *antichain graph* the graph $\Gamma = (\Phi, \prec)$. Let Φ_i be the set of all feasible subsets containing activity A_i.

Let $\sigma = (a_1, \ldots, a_{|\sigma|})$ be a sequence of valid antichains, and for all $A_i \in V$, let $\sigma_i = \Phi_i \cap \sigma$ be the set of antichains of σ containing A_i. σ is said *ordered* if the

order of this sequence respects the partial order given by $\prec$, that is to say $\forall l, l' \in \{1, \ldots, |\sigma|\}, l < l' \Rightarrow a_{l'} \not\prec a_l$. In the following, the positive vector $\mathbf{z} \in \mathbf{R}_+^{|\sigma|}$ will represent the respective amounts of time during which all activities of the feasible subsets of σ are simultaneously executed in the associated schedule, as it is introduced in section 3.2.2.

PROPOSITION 8.1.– *Let σ be a sequence of valid antichains, and $\mathbf{z}$ be a positive vector indexed on σ. The pair $(\sigma, \mathbf{z})$ represents a preemptive schedule if and only if σ is ordered and $\mathbf{z}$ verifies $\forall A_i \in V, \sum_{l \in \sigma_i} z_l = p_i$. Its makespan is $\sum_{l \in \sigma} z_l$.*

By a slight abuse of notations, a pair $(\sigma, \mathbf{z})$ where σ is ordered and $\forall A_i \in V, \sum_{a \in \sigma_i} z_a \geq p_i$ represents also a preemptive schedule, since it can be easily transformed into a regular one.

Mingozzi et al. [MIN 98] introduced the following linear program $\mathcal{P}$ that contains (only) n constraints and $|\Phi|$ variables, except that they efficiently restricted the variable set to the "undominated feasible subsets" (see section 3.2.2).

$$(\mathcal{P}) : \min Z = \mathbf{1} \cdot \mathbf{z}, \quad M\ \mathbf{z} \geq \mathbf{p},\ \mathbf{z} \in \mathbf{R}_+^{|\Phi|}.$$

It is issued from the exact time-indexed mixed integer linear formulation of section 3.2.2, with a much more higher number of variables and constraints. Some constraints of this MIP were relaxed or partially relaxed and the time-indexation was avoided thanks to the introduction of these real-valued "duration" variables $\mathbf{z}$. Hence it is a relaxation, but not only in the sense of the traditional integrality constraint relaxation.

$\mathbf{1}$ is the vector of $|\Phi|$ elements that are equal to 1. The incidence matrix $M(n, |\Phi|)$ is defined as follows: $M_{il} = 1$ if activity A_i belongs to antichain a_l, 0 otherwise. Thus these constraints ensure that for each activity $A_i \in \mathcal{A}$ the sum of the duration of the feasible subsets containing A_i is at least equal to its processing time p_i. The definition of the variables includes the resource and disjunctive constraints, that will be taken into account in the pricing subproblem of the column generation resolution (see next subsection).

We say that an antichain a_l is *active* for a solution $\mathbf{z}$ of $\mathcal{P}$ if and only if $z_l > 0$. Let us denote by $ACT(\mathbf{z})$ the set of active antichains for $\mathbf{z}$. We now define $\mathcal{S}_P$ as the set of solutions $\mathbf{z}$ of $\mathcal{P}$ for which we can build an ordered sequence with exactly all the elements of $ACT(\mathbf{z})$, and state the following theorem.

THEOREM 8.1.– *Solving the preemptive version of the RCPSP consists in finding an element of $\mathcal{S}_P$ that minimizes Z.*

Finally, a necessary and sufficient condition for a solution $\mathbf{z}$ of $\mathcal{P}$ to belong to $\mathcal{S}_P$ is the lack of oriented cycle in the antichain subgraph induced by $ACT(\mathbf{z})$. Note that an oriented sequence can be found through a simple topological order search in this subgraph.

This theoretical formulation leads us to different approaches and algorithms, described in the next subsections.

8.3.2. *A specific neighborhood search algorithm*

8.3.2.1. *Descent approach*

We establish in [DAM 08] the two following theoretical results. The set of the basis solutions (in the Simplex sense) of $\mathcal{P}$ in $\mathcal{S}_P$ is dominant. Furthermore, the set $\mathcal{S}_P$ is connected in the sense that two elements of $\mathcal{S}_P$ are neighbors if and only if one can be obtained from the other by exactly one Simplex pivoting on $\mathcal{P}$. It follows that one may go from any solution of $\mathcal{S}_P$ to any other by a sequence of Simplex pivotings in $\mathcal{P}$.

Thus we devised a descent algorithm based on the Simplex. Indeed, if we have initially a solution $\mathbf{z}^0$ of $\mathcal{S}_P$ (loaded in $\mathcal{P}$ from an initial feasible schedule provided by a simple list algorithm with random priorities), the idea of this algorithm is to check at each step k if the switch of outing and entering variables (resp. z_o and z_e) proposed by the Simplex pivoting will lead to a basis solution $\mathbf{z}^k \backslash \{z_o\} \cup \{z_e\}$ still in $\mathcal{S}_P$, that is to say that the antichain subgraph induced by $ACT(z) \backslash \{a_o\} \cup \{a_e\}$ does not contain any oriented cycle. If this test proves negative, the switch does not occur, and we try another one proposed by the Simplex. Otherwise, the classical Simplex pivoting is computed. The process is iterated until no candidate variable verifies the test: this situation corresponds to a local minimum of the problem. The incremental version of this test in the preemptive case corresponds to the detection of an oriented cycle in the antichain subgraph going through the new element a_e. Note that the non-preemptive version of this test has far strengthener constraints.

As the number of feasible subsets, i.e. the number of variables, is very large (exponential in n) and a few of them are interesting for our problem, we set up a column generation module (see section 3.2.2, without time discretization). Each dual variable π_i is associated to activity $A_i \in V$ and an entering column corresponds to a valid antichain a_e that also has to verify $\sum_{A_i \in a_e} \pi_i > 1$ due to our formulation of $\mathcal{P}$. Hence, the subproblem is a multidimensional knapsack-stable problem, in which each vertex A_i has a unit profit π_i and a multidimensional weight $b_{ik}, R_k \in \mathcal{R}$. It is computed here by a branch-and-bound procedure like in [BRU 00]. The disjunctions (A_i, A_j) are defined by some resource constraint violations ($\exists R_k \in \mathcal{R}, b_{ik} + b_{jk} > B_k$) and the precedence relations between activities ($(A_i, A_j) \in TE$ or $(A_j, A_i) \in TE$), or can be strengthened through constraint propagation techniques

(see [BAP 04], [DEM 05], [BRU 98]), at least each time that the best known upper bound of the makespan is improved. A specific preprocessing filtering step is added on this subproblem, to prevent from building trivial oriented cycles. This preprocessing is removed to check if a local minimum is indeed reached.

8.3.2.2. *Diversification techniques*

Diversification techniques were proposed to get out of the numerous local minima. The idea is to build from each of them a feasible solution for which it has been proven that it can be reloaded in basis of the linear program, in order to reiterate the descent (intensification) process.

The first one, called *reconstruction*, consists in building, from the feasible subset family associated to an optimal solution of the relaxation $\mathcal{P}$, an ordered sequence, allowing some modifications in the subsets. In [MÖH 03], the authors generate such schedules by applying list algorithms with priority lists derived from the order of the jobs in the solution of a Lagrangian relaxation. Here, the following method is inspired from Moukrim and Quilliot [MOU 05], concerning the identical parallel machine scheduling problem. The idea is to build step by step an ordered sequence of feasible subsets. By induction on the feasible subsets of the solution of $\mathcal{P}$ in its initial arbitrary order, we ensure through specific switches of activities that the subsequence from the beginning to the current antichain is an ordered sequence. Because of these switches, the subsets do not forcefully respect the resource constraints first, hence at the end of this induction process, a postprocessing step is performed to make these subsets respect them and the sequence represent a feasible preemptive schedule. The duration variables are provided by the reloading in $\mathcal{P}$. An other reconstruction algorithm has also been designed in order to take into account the associated durations of each feasible subset in the solution of $\mathcal{P}$.

The second diversification algorithm, called *perturbation*, was devised to avoid the fact that during the descent algorithm described above, a valid and Simplex candidate antichain a_e is not allowed to enter the basis antichain family f because of an oriented cycle in the subgraph induced by $f \cup \{a_e\}$. The idea of this algorithm is to force the entering of a_e. Indeed, a_e may be present in “good”solutions, but some antichains of f forbid z_e to enter the basis through only one Simplex pivoting. It reuses the specific switches of activities between the feasible subsets and provides a still ordered sequence of feasible subsets, containing a_e, that do not forcefully respect the resource constraints. The postprocessing step explained above is also performed at the end of this feasible solution perturbation, to build an other more "distant" feasible solution in $\mathcal{S}_P$.

8.3.2.3. *Experimental results*

The experiments were computed on a PC Duron 1GHz - 256 Mb RAM. The results of this specific neighborhood method (descents with 20 reconstructions and 5 perturbations between each reconstruction) on the PSPLIB instances (see [KOL 97] and

chapter 7) with 30 activities, called $KSD30$, are very good, since the optimum was obtained for 442 over 480 instances. The average and maximum gaps are respectively 0.15 % and 5.41 %, in an average time of 9.9 s. Note that the optimal value is obtained for these instances thanks to the branch-and-bound algorithm described in the following section. This is not the case for the instances with 60 activities, called $KSD60$, but we can assume that the experiments are still very good on them, since the average gap with the lower bound given by the solution of $\mathcal{P}$ is similar to the one obtained on $KSD30$ (respectively 2.38 % and 2.15 %). But for them the solutions are obtained within an average time of 103.4 s.

8.3.3. *Exact methods*

8.3.3.1. *Branch-and-bound*

A branch-and-bound algorithm derived from the formulation of section 8.3.1 based on $\mathcal{P}$ has been implemented and presented in [DAM 06b]. It was devised first to improve our best known preemptive lower and upper bounds, and eventually to find the optimal preemptive solutions of the KSD30 instances. The bounding process consists in solving $\mathcal{P}$ (with a few additional simple logical constraints) to get a local lower bound $\overline{\mathbf{z}}$ of the optimal makespan, whereas the branching process consists in forbidding the variables corresponding to the vertices of an oriented cycle in the antichain subgraph induced by $ACT(\overline{\mathbf{z}})$.

Indeed, for each antichain a_l of such an identified oriented cycle, the current node of the search tree derives one subproblem in which the corresponding variable z_l is not allowed to appear in the associated solution of $\mathcal{P}$. Hence, for any subproblem in the search tree, there is a list $\mathcal{L}$ of forbidden feasible subsets. To an algorithmic point of view, this is done by setting the corresponding duration variables $z_l, a_l \in \mathcal{L}$, to 0. If such a solution does not contain any circuit, then it corresponds to a feasible preemptive solution. In a best-first exploration of the search tree, the process is stopped since this solution has the minimum value, that is to say the minimum makespan.

The branching process has been first improved by only considering the oriented cycles of minimal length in the antichain subgraph. We proved that it is sufficient in our research only to consider the chordless oriented cycles, which are easier to identify. Furthermore, the idea was then to detect more precisely the activities involved in these antichain cycles. The forbidden list $\mathcal{L}$ became a list of forbidden couples of activities, that is to say disjunctions. This latter point is of course implicitly well treated in our column generation process, as the constraints deal with disjunctions of activities.

Finally, one may observe in our very first experiments that the main chordless oriented cycles are of length 2 for the KSD30 instances. Thus we proposed three filtering steps in the branching process, in order to add even more disjunctions, so

to forbid even more feasible subsets in the numerous subproblems of the search tree. One of these filtering steps has provided all optimal preemptive solutions of $KSD30$ in an average (resp. maximum) time of 11 minutes (resp. 9 hours) and an average (resp. maximum) number of treated nodes of 1050 (resp. 25717).

8.3.3.2. *Branch and Cut and Price*

Another idea to use this formulation in an exact resolution is to build a Mixed Integer Linear Program (MILP) [DAM 05] [DAM 06a] and to use the recent solving techniques and frameworks dedicated to it. Indeed, if we add a binary *presence* variable for each feasible subset in $\mathcal{P}$ which is equal to 1 when the feasible subset is chosen in the solution and 0 otherwise, it is easy to formulate the non-circuit constraints. This provides a MILP with an exponential number of variables (two for each feasible subset) and an exponential number of constraints (one for each activity like in $\mathcal{P}$, plus one for each oriented cycle in the global antichain graph $(\Phi, \prec)$, plus one for each feasible subset, which is called the *Big-M coupling* constraint). But these numbers of variables and constraints are far less large than for the other classical time-indexed formulations, in particular the one of Mingozzi et al. [MIN 98]. Such works are also likely to be done when not all activities are preemptive, and naturally for the non preemptive case of the RCPSP.

8.4. Conclusion

In this chapter, we have focused on the high quality and promising results of recent works for the resource constrained project scheduling problem, including its rational preemptive version. Our approach deals with a new formulation of this problem involving the feasible subsets of activities, but an important point is that it is not time-indexed and that the underlying linear program is quite easy to solve. This leads to several types of algorithms, such as an original generic iterated descent algorithm, and such as exact methods that provide very good and original experimental results.

One of the latter methods deals with the very efficient and promising *Branch and Cut and Price* techniques, for which the implementation and experiments are performed at the present time and are likely to provide really interesting and good results for RCPSP.

But other interesting original theoretical results can derive from these approaches. Like for the parallel machine scheduling problem, a class of instances can be found, for which the value of the optimal solution of $\mathcal{P}$ is equal to the makespan of the optimal preemptive solution of the problem. A polynomial algorithm can also be devised to build an optimal schedule from any solution of $\mathcal{P}$, called *optimal reconstruction*.

Chapter 9

Multi-Mode and Multi-Skill Project Scheduling Problem

9.1. Introduction

This chapter deals with two different problems: the Multi-Mode Resource-Constrained Project Scheduling Problem (MM-RCPSP) and the Multi-Skill Project Scheduling Problem (MSPSP). What is particular about those two problems is that resources can be assigned according to different ways, eventually impacting processing time of activities. For the first problem, activities may be processed according to several ways, that means different amounts of resources required and corresponding processing time. For the second one, resources are able to be assigned to different kinds of resource requirements. This major difference with traditional RCPSP, that can be called *resource flexibility*, leads to a significant increasing of the number of feasible solutions. To build a solution we have to fix (1) the way the activities have to be processed and (2) the starting times of the activities. The latter problem is then equivalent to solving a traditional RCPSP instance.

Section 9.2 deals with the MM-RCPSP, and section 9.3 with the MSPSP. Each section draws the specificities of its studied problem based on the presentation of a simple example, and then resolution methods are described. We focus on branch-and-bound methods (see Chapter 5), because those ones are significantly different than traditional branch-and-bound for the RCPSP, exhibiting the structural difference between these problems and the RCPSP. For the Multi-Mode Resource Constrained Project Scheduling Problem, exact methods are based on partial schedules and at each

---

Chapter written by Odile BELLENGUEZ-MORINEAU, Emmanuel NÉRON.

node of the search tree both mode of execution and starting time are fixed for an activity while for the Multi-Skill Project Scheduling Problem, the branching scheme is based on time-windows reduction and the assignment problem is solved when a leaf node is reached.

9.2. Multi-Mode RCPSP

9.2.1. *Problem presentation*

The Multi-Mode Resource-Constrained Project Scheduling Problem (MM-RCPSP) is an extension of the traditional Resource-Constrained Project Scheduling Problem, introduced by [TAL 82]. There is a project made up of a set of activities, linked by precedence relations, that are traditionally represented using an activity-on-node graph. Resources considered are renewable or non-renewable resources. Nevertheless, there exists a main difference in the Resource Constrained Project Scheduling Problem with multiple modes of execution: for any activity of the project there may exist several *modes* of execution. A mode of execution corresponds to a given amount of each resource and a fixed processing time. Thus, each mode corresponds to a trade-off between resource consumptions and processing time of the activity.

The aim is to build a schedule that minimizes the total duration of the project, according to precedence and resource constraints. A schedule is defined by both a starting time for any activity of the project, and its associated mode, that determines its resource consumptions and its processing time. According to the fact that the traditional RCPSP is $\mathcal{NP}$-hard in the strong sense [BLA 83], this problem with multiple modes of execution is clearly $\mathcal{NP}$-hard in the strong sense. Many methods have been proposed to solved this problem (see [BRU 99] for a literature review).

Notation used are the same as those used for the traditional RCPSP. In addition, let us denotes the following:

Notation	
M_i	number of modes that exist for A_i. M_0 and M_{n+1} are equal to zero.
$p_{i,m}$	processing time of the activity A_i executed in mode $m, 1 \leq m \leq M_i$.
$b_{i,m,k}$	amount of resource R_k used during the execution of A_i in mode $m, 1 \leq m \leq M_i$.

Table 9.1. *Notation for Multi-Mode Resource-Constrained Project Scheduling Problem*

Here is an example of this problem: we consider the project defined by the precedence graph presented on the left part of Figure 9.1. This project requires three renewable resources: R_1, R_2, R_3, with the following capacity: $B_1 = 1, B_2 = 2, B_3 = 2$. Requirements of activities according to the different modes are presented in Table 9.2. A feasible solution for this instance is also presented in Figure 9.1.

Activities	A_1		A_2			A_3		A_4			A_5
M_i	2		3			2		3			1
	$b_{i,1,k}$	$b_{i,2,k}$	$b_{i,1,k}$	$b_{i,2,k}$	$b_{i,3,k}$	$b_{i,1,k}$	$b_{i,2,k}$	$b_{i,1,k}$	$b_{i,2,k}$	$b_{i,3,k}$	$b_{i,1,k}$
R_1	1	1	1	1	-	1	-	-	1	-	-
R_2	2	1	1	1	1	-	-	2	2	1	-
R_3	-	-	1	-	-	2	1	-	-	1	2
$p_{i,m}$	3	5	2	4	7	4	8	4	5	5	3

Table 9.2. *Example of mode requirements of activities*

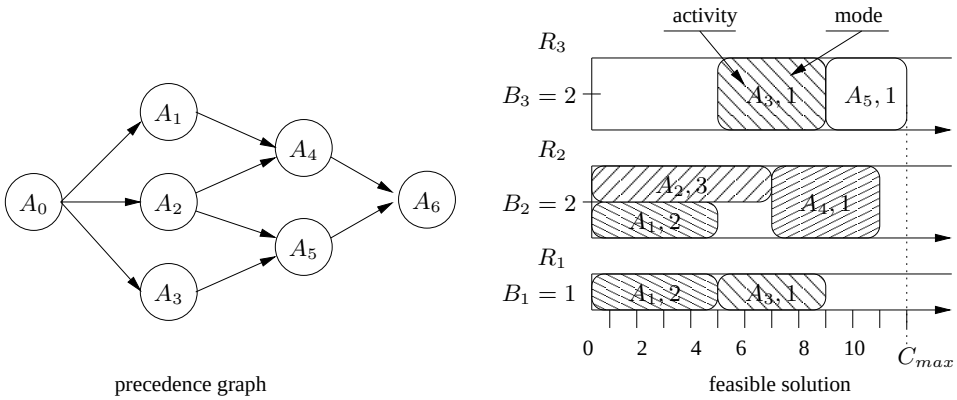


Figure 9.1. *Example of precedence graph and corresponding feasible solution*

Notice that some modes may be identified as *unfeasible* or *inefficient*. A mode is called unfeasible if it requires an amount of resources greater than what is available. A mode is considered to be inefficient if there exists another mode for the same activity that requires an amount equal or smaller of each resource and that has a smaller processing time. In the example presented in Table 9.2, mode 2 of A_4 is inefficient because mode 1 requires the same amount of resource R_2, less resource R_1 and has a smaller processing time.

This problem has been extensively studied. Many studies focus on heuristics and meta-heuristics for solving medium and large size instances, for instance using simulated annealing [BOC 96] or evolutionary algorithm [HAR 98a], [MOR 97], [ÖZD 99]. Recently, lower bounds based on Integer Linear Programming formulation have been proposed by [BRU 03]. Here we present some key points of exacts methods [SPR 94], [HAR 98b] including branching scheme, lower bounds and dominance rules.

9.2.2. *Branching schemes for solving multi-mode RCPSP*

Several exact methods have been proposed to solve this problem. Inspired from methods designed for solving RCPSP (see Chapter 5), most of those methods build partial schedules. At each node of the search tree an activity (or a subset of activities) is scheduled, i.e., a mode of execution and a starting time are fixed. Resources have to be assigned according to the chosen mode of execution and the processing time can be deduced. A leaf node is reached when all activities are scheduled, i.e., a global solution has been built, respecting resource and precedence constraints.

Here we present two branching schemes. In the method proposed by [SPR 94], only one activity is fixed at each node of the search tree. At a given node, we consider the set of eligible activities, i.e., those whose all the predecessors are scheduled. For each possible mode of execution of each eligible activity a node is added to the search tree. Thus, in a node the mode of execution of an activity is chosen. This node is pruned if there is no feasible starting time according to the time-window of the activity (time-windows are computed according to the best known solution, the precedence relations and the current partial schedule) and according to the resource constraint. Otherwise, the activity is scheduled at its earliest feasible starting time, in order to build semi-active schedule. In the example presented in Figure 9.1, if we consider a partial schedule where A_1 and A_2 are scheduled, activity A_3 is scheduled at $t = 5$, which is the earliest starting time that does not lead to a resource conflict. It is possible only in the case that it does not violate the time-windows of A_3, otherwise the node is pruned.

The method presented by [HAR 98b] schedules several activities at a time. This branching scheme may be seen as an extension of the branching scheme based on delaying alternative proposed by Christophides et al. [CHR 87] and Demeleumeester and Herroelen for solving an RCPSP [DEM 92b]. At a given node, the earliest completion time of all activities in progress in the partial schedule is computed. At this time-point, the set of *mode alternatives* is enumerated. It corresponds to all the possible choices of modes to execute the eligible activities. Thus, a mode alternative fixes a mode for each activity. According to the modes chosen for every activities and the resource consumptions, it may lead to a resource conflict. Thus, the set of *mode delay alternatives* has to be enumerated. Associated to a mode alternative, a mode

delay alternative corresponds to a minimum set of activities that have to be delayed to solve the resource conflict detected. Thus, the branching scheme of this method is based on the enumeration of all mode delay alternatives at each stage. In the example presented in Table 9.2, at $t = 0$, A_1, A_2 and A_3 may be scheduled. The first set of mode alternative corresponds to schedule those activities in their first mode, so it may be defined by: $\{A_1, 1; A_2, 1; A_3, 1\}$, where any chosen activity is directly followed by the number of its chosen mode of execution. This alternative leads to a resource conflict. According to the fact that no pair of activities can be executed in parallel without leading to a resource conflict, the distinct sets of mode delay alternative are: $\{A_1, 1; A_2, 1\}, \{A_1, 1; A_3, 1\}, \{A_2, 1; A_3, 1\}$. Thus, the created nodes, corresponding to the first set of mode alternative, are : $\{A_3, 1\}$, $\{A_2, 1\}$, $\{A_1, 1\}$.

The strategy to explore the search tree is chosen according to a priority rule combined with a traditional depth-first strategy. This order may really change the efficiency of the method. So, the activity are either sorted by the length of their minimum processing time, maximum processing time, or even average processing time. Then for a given activity, modes are sorted according to an associated weight. This weight is inversely proportional to the processing time associated to the mode: a mode with a heavy weight is chosen first because it corresponds to a small processing time.

9.2.3. *Dominance rules*

Dominance rules have been proposed in order to reduce the search tree that may become huge depending on the number of modes and the number of activities to consider [SPR 94], [HAR 98a]. Here we present a brief overview of such rules.

1) If an eligible activity cannot be scheduled without violating its latest finishing time (computed according to the best known solution and precedence relations), in any mode, then the corresponding node can be pruned.

2) If, in a partial schedule, an activity can be left-shifted without violating neither the resource nor precedence constraints, then the node corresponding to this partial schedule can be pruned.

3) In a partial schedule, if no eligible activity can be scheduled before the end of an activity A_i, executed in mode m, and if A_i may be executed in mode m' with a shorter processing time without violating resource constraints, then the node corresponding to the schedule of A_i in mode m may be pruned.

4) If there exists a partial schedule where an activity A_i and an activity A_j are added respectively in mode m_i and m_j with the same starting time, it is not necessary to explore the node where, in the same partial schedule, activity A_j in mode m_j and then activity A_i in mode m_i are added because they also will have the same starting time, so the partial schedules are equivalent.

5) Consider that there exists a first partial schedule PS where an activity A_i in mode m_i and an activity A_j in mode m_j are added to get a first partial schedule PS^1.

Assume that the completion time of A_i in this first schedule, C_i^1, is equal to the starting time of A_j, S_j^1. If, in another node, activities A_j in mode m_j and A_i in mode m_i are added to the same partial schedule PS to get PS^2, with C_j^2=S_i^2 and if the first activity not yet scheduled can not be scheduled earlier in PS^2 than in PS^1, then the second partial schedule PS^2 should not be built, the corresponding node has to be pruned.

6) If a partial schedule PS has a completion time C, and if in another node there exists a partial schedule containing the same activities that allows the next activity to start at a time t equal to or greater than C, the node corresponding to this second partial schedule should not be explored.

This extension of the traditional RCPSP allows taking into account time-cost trade-off. Thus, it may be used to represent many real problems. However, all resources considered are one-purpose, i.e., they may satisfy only one kind of resource requirement. This is why another extension has been introduced: the Multi-Skill Project Scheduling Problem. In this problem, resources correspond to persons that master a subset of the skills required by the different activities of the project.

9.3. Multi-Skill Project Scheduling Problem

The Multi-Skill Project Scheduling Problem has been proposed by Néron and Baptista [NÉR 02]. The main feature of this model is to consider resources that are staff members, thus renewable and disjunctive, mastering several skills. Initially it has been proposed to model project scheduling problems in IT companies.

9.3.1. *Problem presentation*

MSPSP can be seen as an extension of traditional RCPSP. In the Multi-Skill Project Scheduling Problem, the project to schedule is traditionally made up of a set of activities, linked by precedence relations. The main difference is the resource considered and the way to define resource requirements: resources are staff members assigned to the project. Thus, resources considered are renewable and disjunctive, i.e., a staff member can be assigned at most to one requirement at a time. The way these resources are used is a consequence of the skills they master among all those taken into account for the project. Thus, for each person in the staff, we exactly know which skill he/she masters. Symmetrically, requirements of activities are defined in terms of skill requirements, i.e., the number of persons for each skill required to process the activity. Thus, the staff members assigned to the execution of an activity are a subset of persons able to satisfy its skill requirements.

The goal is to build a schedule to minimize the total project duration. This schedule has to respect precedence and resource and skill constraints. According to the different

skills every person masters, there may exist many possible subsets of persons able to satisfy the skill requirements of the activities. Thus, when the starting time of each activity is fixed, there may correspond many feasible schedules. Nevertheless, a valid schedule has to respect the following constraints: a person cannot be assigned to more than one unit of skill requirement at a time and not to a requirement corresponding to a skill he/she does not master. Moreover, all the persons who start an activity have to be assigned to it all along its processing. Furthermore, the model used also takes into account unavailability periods of the resources. Thus, a person cannot be assigned to the execution of an activity during a period he/she is not available. According to the fact that the traditional RCPSP is $\mathcal{NP}$-hard in the strong sense, this problem is also $\mathcal{NP}$-hard in the strong sense. Notations used are based on those proposed for the traditional RCPSP, and the following ones are added (see Table 9.3).

Notation	
$\mathcal{L} = \{L_1, ..., L_p\}$	set of skills considered for the project.
$\mathcal{R} = \{R_1, ..., R_q\}$	set of resources, i.e., persons assigned to the project.
	(capacity B_k of a person is always equal to 1.)
$b_{i,h}$	number of persons required to do the skill L_h
	all along the execution of the activity A_i.
$MS_{k,h} = 1$	if skill L_h is mastered by person R_k, 0 otherwise.
$Disp(R_k, t) = 1$	if R_k is available between, t and $t + 1$, 0 otherwise.

Table 9.3. *Notation for Multi-Skill Project Scheduling Problem*

An instance of this problem is presented in Figure 9.2. Activities are submitted to precedence relations, modelled using an activity-on-node graph. For each activity, a processing time and a fixed number of persons mastering skills are given. For instance, activity A_1, needs 1 person doing skill L_1, 1 person doing skill L_2 and 2 persons doing skill L_3. The subset of skills that are mastered by staff members are also given. For instance $R_2(L_1, L_2, -)$ means that resource R_2 cannot be used to do skill L_3. The Gantt diagram shows a feasible solution for this instance, i.e., respecting precedence constraints, resource and skill constraints and unavailability periods.

In the model presented in this chapter, we only consider that a person masters a skill or not. However, in some real-life cases, it happens that there exist hierarchical levels for each skill, in order to take into account that some specific tasks have to be done by an experienced employee [BEL 05]. That means for every activity, the manager defines the level of difficulty, i.e., the level of mastering required to satisfy needs. By the same way, the maximum level each employee masters for each skill is known. This notion of hierarchical levels of skills has been proposed for problems of employees assignment by [TOR 03]. However, this particular case may be represented

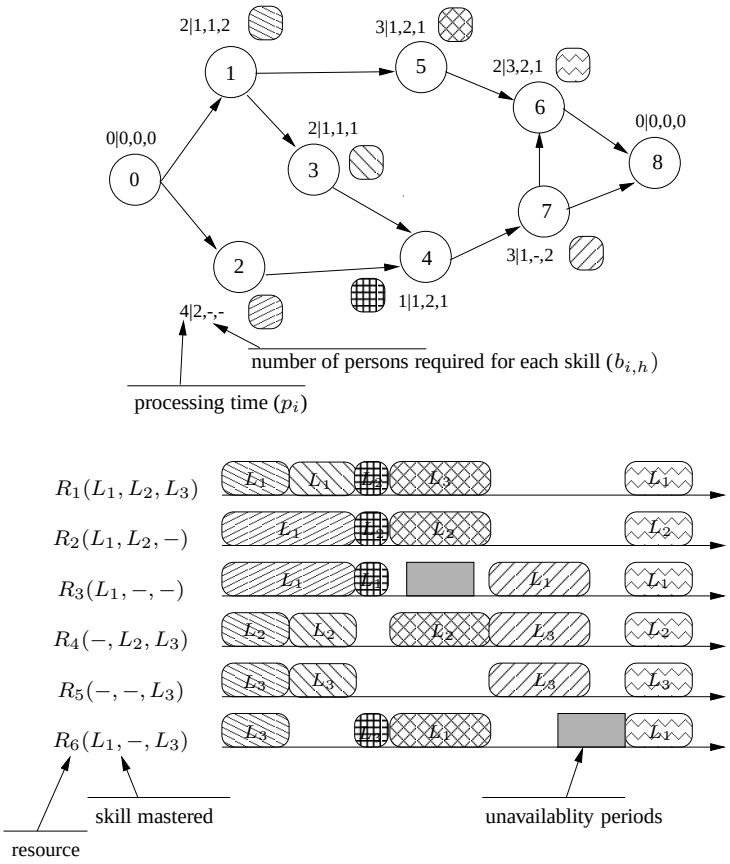


Figure 9.2. *Example of instance and corresponding feasible solution*

by the model presented above: a dummy skill is added for each specific level of each skill, and an employee masters all the dummy skills corresponding to a level equal to or lesser than the larger level he/she masters.

The Multi-Skill Project Scheduling Problem may also be seen as a Multi-Mode RCPSP. In this case, modes of execution of a given activity have the same processing time, and resource requirements correspond to a given subset of persons that may be assigned to the activity. Thus, we have to enumerate for each activity all feasible subsets of employees that can satisfy requirements, each of these subsets corresponds to one mode in the multi-mode context. However, according to the different skills that each person masters and the number of persons required to process an activity, the

number of modes may be huge. To add to this, as described in section 9.2.2, main exact methods for solving the Multi-Mode Resource Constrained Project Scheduling Problem are based on an enumeration of the activity modes, and this is the reason why those methods are not efficient to solve the Multi-Skill Project Scheduling Problem.

The fact that resources can be used to satisfy more than one kind of resource requirements has already been proposed in the context of shop floor scheduling using the notion of multi-purpose machines (MPM) and multi-processor tasks (MPT). Compared to the MPM problem, the main difference is that activity, in MSPSP, may require more than one resource to be processed, these resources have to be picked up depending on the skill associated to resources. A mixture of MPM [JUR 92] and MPT [KRÄ 95] is used in Flexible Job-Shop [DAU 96]. For this problem, authors have proposed heuristic methods. Here we present a branch-and-bound method for solving exactly the Multi-Skill Project Scheduling Problem.

9.3.2. *Branching scheme based on time-windows reduction*

Among the methods used to solve this problem, there exists a branch-and-bound [BEL 07]. The branching scheme used is based on the reduction of the slack of the activities. This has been inspired by [CAR 91] (see section 5.2.2). First of all, time-windows, $[ES_i, LC_i]$, of activities have to be computed, according to the best known upper bound, the precedence graph and the partial solution. The best known solution is initially determined using a heuristic method, and it is updated each time a better solution is reached in the search tree. At each node of the search tree, once every time-windows are computed, an activity A_i is chosen, and two nodes are created: in the first one, its deadline, LC_i, is decreased, in the second one its release date, ES_i, is increased. These reductions are equal to an half of the slack ($LC_i - ES_i - p_i$) of the activity, such that the set of feasible starting times for the A_i is split into two disjoint subsets.

Moreover, at each node of the search tree, at least one time-window is reduced, (some others may be reduced too, due to the propagation on precedence graph) so it necessary leads to leaf nodes, where every activities have a slack equal to 0. Thus, starting times and completion times of activities are fixed. Nevertheless, the assignment problem, deciding who is assigned to satisfy skill requirement of activities, is not solved yet, so a leaf node may correspond to several valid schedules or to no one. Such a problem that corresponds to the MSPSP instance presented in Figure 9.2 is presented in Figure 9.3.

Determining if there exists a feasible assignment of persons to fixed activities respecting skill constraints can be seen as a Fixed Job Scheduling Problem, which is $\mathcal{NP}$-Complete in the strong sense [KOL 91]. Thus, each leaf node is solved using

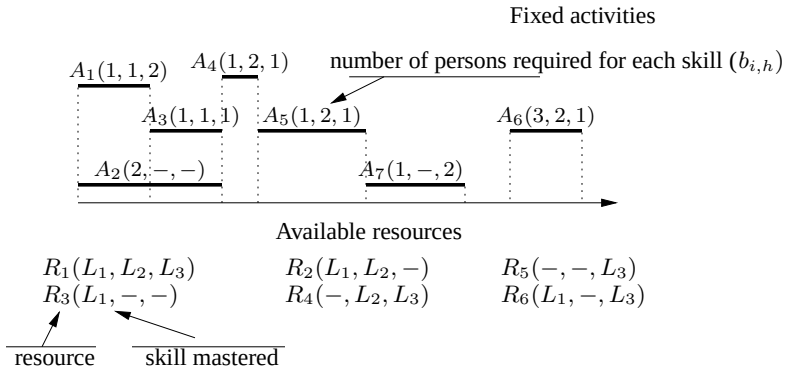


Figure 9.3. *A leaf of the initial search tree : a Fixed Job Scheduling Problem instance*

an integer linear programming formulation, in order to prune the node if it is unfeasible, or to find a solution if it exists. In this latter case, the solution to this assignment problem with fixed starting time is a solution for the global problem.

9.3.3. *Lower bounds and time-bound adjustments*

During the search, each node is evaluated by lower bounds in order to know if the node may lead to a solution better than the best known one or if it has to be pruned. Two complementary methods are used for this evaluation. Those two methods are destructive ones. That means a value D is fixed as an upper bound and tests are performed to detect a contradiction. D is equal to the makespan of the best known solution minus one. If D cannot be respected as a global deadline, then the node cannot lead to a solution strictly better than the best one, so it can be pruned.

The first method used to detect contradiction has been proposed by Bellenguez and Néron [BEL 05]. It is inspired from [BAP 99], [NÉR 01]. It is based on energetic reasoning. On a given time-interval, once every time-windows $[ES_i, LC_i(D)]$ of activities are computed, it is necessary to compute the mandatory part of the activities. The mandatory part $w(i,t_1,t_2)$ of an activity A_i on an interval $[t_1;t_2]$ is the minimum part of the activity that has to be processed on this interval to respect the time-window of the activity and is computed:
$w(i,t_1,t_2) = min(max(0, ES_i + p_i - t_1), max(0, t_2 - (LC_i(D) - p_i)), p_i, t_2 - t_1)$.

PROPOSITION 9.1.– *If there exists a time-interval $[t_1;t_2[$ where the assignment problem of the mandatory parts of activity to staff members, respecting skills constraints has no solution according to the resource constraint, then there exists no solution that respects the global deadline D.*

Notice that this assignment problem can be solved in polynomial time.

The second lower bound is based on the graph of antichains, introduced for the RCPSP by [MIN 98]. This method is based on pairs of activity that may be in progress at the same time, without violating neither the precedence constraints nor the resource constraints. Thus, for each pair, it is necessary to check (1) if the two activities are not linked by a precedence relation, (2) if their time-windows overlap and (3) if their execution in parallel does not lead to a resource conflict. The compatibility graph is made up of a node per activity, and two activities are linked if they can be in progress at the same time, i.e., satisfying (1) (2) and (3). Weights associated with nodes are equal to the duration of the corresponding activity. In this graph, a maximum weighted stable set is computed, exactly using an ILP formulation, or heuristically using a greedy algorithm. The computed stable set corresponds to a subset of activities that cannot be in progress at the same time. Thus, if its weight, i.e., the sum of the processing times of the activities belonging to the stable set, is greater than D, then there exists no solution that respects the global deadline D [BEL 05].

In order to reduce the number of stages required to reach a leaf node, it is useful to adjust the time-windows of activities. Thus, time-bound adjustments presented by [BAP 99] for traditional RCPSP have been adapted. Due to their time-consumption they are performed only at the root node. In other nodes, same time-bound adjustments are computed but on a relaxation of the problem, where the constraint ensuring that a person cannot do several skills at the same time is relaxed, so the problem is equivalent to a traditional RCPSP with cumulative resources. For each of those resource, which corresponds to a specific skill, the capacity is equal to the number of persons that master the corresponding skill [BEL 07].

9.3.4. *Dominance rule*

Added to the lower bounds, there exists another way to limit the number of nodes. At a given node, it is possible to compute the *central mandatory part of an activity*, [HAO 03], which corresponds to the time-slots during which an activity must be in progress to respect its time-window (see Figure 9.4). Using those central mandatory parts of activities, we define dummy activities with a starting time equal to the starting time of the central mandatory part and a processing time equal to the length of the central mandatory part. Thus, it defines an assignment problem with fixed starting times. This one can be solved using the same method as the one used for a leaf node (see Figure 9.3). If this problem has no solution, central mandatory parts cannot be scheduled, then it is not possible to schedule the whole activities, so the corresponding node will not lead to a feasible solution, and can be pruned.

This dominance rule is highly time-consuming and is not really efficient if the time-windows of activities are not enough strengthened, only few contradiction can be deduced, this is the reason why it is not used at each node.

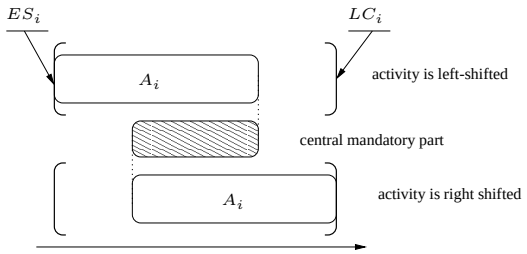


Figure 9.4. *Central mandatory part of any activity A_i.*

9.4. Conclusion and research directions

In this chapter two extensions of traditional RCPSP have been presented. Both are linked to resource flexibility. In the traditional RCPSP, resources needed and amounts of these resources required to process activities are known. In these two extensions, there exist alternative resource requirements for activities. The main difference lies in the fact that deciding what resource will be used and what amount of this resource is required, are decision variables that must be fixed during the resolution process. Thus, resolution methods, especially exact methods, are much more complex.

Even if these two models, namely Multi-Mode RCPSP and Multi-Skill Project Scheduling Problem are much harder to solve in practice than traditional RCPSP, their practical interest is obvious: these extensions mixing project definition and resource flexibility to process activity are one of the most promising research directions to solve real-life applications. Thus, the huge amount of work in the literature dedicated to heuristics for these problems would be used to obtain good solutions. Nevertheless, lower bounds, time-bound adjustments and also exact methods can be used to solve heuristically large size instances, using large neighborhood search for instance (see Chapter 6).

Chapter 10

Project Scheduling with Production and Consumption of Resources: How to Build Schedules

Scheduling problems with resource constraints are NP-hard. Thus, methods are required for enumerating solutions. One method for the RCPSP is based on the notion of arbitrage, which is a preorder on the starting times of activities. An earliest schedule can be computed using the strict order algorithm. We show here how to compute an earliest schedule using AND/OR precedence relations. We also generalize the notion of arbitrage to the more general case of activities producing and consuming a non-renewable resource.

10.1. Introduction

Project scheduling has attracted an increasingly growing interest in recent years, from both a scientific and a practical point of view, because it is an issue in almost every area of the economy. We talk about optimizing codes in computer science, the job-shop problem in industry, timetable development in administration and project planning in construction. Project scheduling is concerned with single-item or small batch production where scarce resources have to be allocated to dependent activities over time. Over the last 10 years scheduling problems with makespan minimization have been the subject of numerous papers. However, the only types of resources usually considered are *renewable* and *non-renewable*. Renewable resources are required

Chapter written by Jacques CARLIER, Aziz MOUKRIM and Huang XU.

by activities at their starting times and given back at their completion times. Renewable means that a pre-specified number of units of a resource are available for every period of the planning horizon. One example of a renewable resource is the workforce. Non-renewable means that a number of units of a resource become available at some time and then can be spent over the remaining planning horizon. Money is an example of a non-renewable resource. The project scheduling problem with non-renewable resources can be solved by the shifting algorithm [CAR 84], which calculates the latest schedules in polynomial time. Solutions to the RCPSP can be enumerated via the notion of arbitrage using the strict order algorithm. We also show here how to compute these solutions using AND/OR precedence constraints. Nowadays, as demands from the industrial sector increase, the integration of these two types of resources is becoming crucial. Laborie [LAB 03b] put forward the concept of a Resource Temporal Network (RTN), which has a great expressive power and in which most of the resources used in scheduling can be modeled. Bouly *et al.* [BOU 04] developed a model which allows resource production by tasks, and provided algorithms to solve the problem with makespan minimization. Here we also discuss the project scheduling problem with production and consumption of resources.

This chapter is composed of four sections. In section 10.2 we recall the graph representation of a project scheduling problem with AND/OR precedence constraints. section 10.3 will be devoted to the RCPSP, and section 10.4 to the project scheduling problem in which activities can produce and consume resources.

10.2. The precedence-constrained project scheduling problem

A project consists of a number of activities or tasks that have processing times and that need to be executed according to a set of precedence constraints. We begin by making the distinction between traditional precedence constraints and AND/OR precedence constraints.

10.2.1. *Traditional precedence constraints*

The traditional precedence constraint is the AND precedence constraint. Each activity may have several predecessors and cannot start until all its predecessors are completed. A project network is given by a directed graph $G = (V, E, v)$ where nodes in V model activities and arcs in E model precedence relations.

The classical objective is the makespan of the project which has to be minimized. Let us assume that the project consists of a set $V = \{A_0, A_1, ..., A_{n+1}\}$ of activities where activity A_0 (respectively A_{n+1}) is a fictitious beginning (respectively termination) activity. It is represented by a directed graph $G = (V, E, v)$. V is the node-set, $|V| = n+2$. E is the arc-set, $|E| = m$. v_{ij} is the length of the arc from A_i to A_j. The

problem is to find the earliest starting time S_j, for all $A_j \in V$, satisfying the following conditions:

1) $S_j \geq max\{S_i + v_{ij} | (A_i, A_j) \in E\}$

2) $S_j \geq 0$, for all $A_j \in V$

To solve the problem there exist several efficient algorithms, such as Bellman's algorithm with a complexity of $O(m)$ when there is no directed cycle, Dijkstra's algorithm with a complexity of $O(n^2)$ when the values are non-negative, and the Floyd-Warshall algorithm [LAW 76], with a complexity of $O(n^3)$ in the general case.

10.2.2. *AND/OR precedence constraints*

Basically, an AND/OR-network is a directed graph (or *digraph* for short). In contrast to standard digraphs, an AND/OR-network has two different sorts of nodes, AND- and OR-nodes, which represent the following constraints. An AND-node is simply a normal node, meaning that the job it represents can be processed as soon as every job preceding it in the graph has been completed. An OR-node, on the other hand, can be scheduled as soon as at least one of its predecessors has completed, and thus represents a constraint. These generalized precedence constraints provide a good model for a variety of applications. The input of the scheduling problem under consideration is a directed graph $G = (V, E, v)$ with nodes of both types, V is the node-set, $|V| = n + 2$. E is the arc-set, $|E| = m$ and v_{ij} is the arc length from A_i to A_j, which models their time lags. We assume that $V = A \cup O$, A being the set of AND-nodes and O the set of OR-nodes.

The problem is to find the earliest starting time S_j, for all $A_j \in V$, satisfying the following conditions:

1) $S_j \geq max\{S_i + v_{ij} | (A_i, A_j) \in E\}, \forall A_j \in A$

2) $S_j \geq min\{S_i + v_{ij} | (A_i, A_j) \in E\}, \forall A_j \in O$

3) $S_j \geq 0$, for all $A_j \in V$

It is easy to see that the problem is transformed into the critical path problem if the set O is empty, and the shortest-path problem if the set A is empty.

In the case where time lags are strictly positive, Möhring *et al.* [MÖH 04] have proposed an $O(|O|log|O|+|E|+|V|)$ algorithm to calculate the earliest starting times. Where there are arcs with zero values they have proposed an $O(|V| + |E| \cdot |O|)$ algorithm. Adelson-Velsky and Levner [ADE 02] have also proposed an $O(|V|+|E|\cdot|O|)$ algorithm, a generalization of Dijkstra's algorithm, to calculate the earliest starting times in this situation. Where negative arcs are allowed, pseudopolynomial algorithms can easily be obtained, but no-one has yet proposed a polynomial algorithm.

10.3. The resource-constrained project scheduling problem

The basic RCPSP can be presented as follows. Assume that the project consists of a set $V = \{A_0, A_1, ..., A_{n+1}\}$ of activities where activity A_0 (respectively A_{n+1}) is a fictitious beginning (respectively termination) activity. Each activity A_i of this set has a processing time p_i. Preemption is not allowed. Each activity A_i needs a certain number of units of resources in order to start its execution. a_{i_k} defines the number of units of resource R_k necessary for activity A_i, and B_k defines the number of units of resource R_k available. S_i is the starting time of activity A_i, while C_i is the completion time of activity A_i. The objective is to find a schedule with minimal makespan that meets the constraints imposed by the precedence relations and by limited resource availabilities.

In order to illustrate the presentation we use the following example.

EXAMPLE 10.1.– Let $V = \{A_0, A_1, A_2, A_3, A_4, A_5\}$ be the set of activities, where activity A_0 (respectively A_5) is a fictitious beginning (respectively termination) activity. There is only one resource $R = \{R_1\}$, $B_1 = 5$. $a_{1_1} = 2, a_{2_1} = 1, a_{3_1} = 3, a_{4_1} = 2, p_1 = 3, p_2 = 2, p_3 = 3$ and $p_4 = 3$. Activity A_1 precedes activity A_3.

10.3.1. *Graph representation*

We use an activity-on-node (AoN) representation of the RCPSP with a directed graph G. The graph $G = (V, E, p)$ is assumed to be without directed cycles and it has nodes in V as activities and arcs in E as precedence constraints. For all $(A_i, A_j) \in E$, $S_j \geq S_i + p_i$. The digraph resulting from Example 10.1 is shown in Figure 10.1.

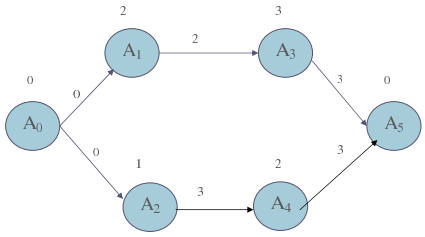


Figure 10.1. *The graph resulting from Example 10.1.*

10.3.2. *The strict order algorithm*

The strict order algorithm [CAR 84] is an algorithm designed for makespan minimization of RCPSP. It takes a complete and compatible arbitrage as input[1] . Then it finds a schedule which is optimal for the defined preorder of activity starting times if and only if the selected arbitrage is complete and compatible.

DEFINITION 10.1.– *An arbitrage of resource R_k is a set of conjunctive arcs $\alpha_k = \{(i_1, i_2), ..., (i_{p-1}, i_p)\}$ with null values, while $\Pi = (i_1, i_2, ..., i_p)$ is a permutation on the set of activities requiring this resource.*

DEFINITION 10.2.– *A complete arbitrage $\alpha = \cup_{k \in K} \alpha_k$ is a union of arbitrages of resources.*

DEFINITION 10.3.– *A complete arbitrage is compatible if there exists a schedule such that for each arc (i, j) in α, $S_j \geq S_i$.*

The strict order algorithm is $O(n^2)$ [CAR 84]. It calculates an earliest schedule which respects a given preorder (Figure 10.2). Note that for efficiency, the algorithm does not necessarily respects the start time order set by the arbitrage and is thus identical to the serial scheduling scheme (see Chapter 6).

1) **Initialization**
$t = 0$;
2) **Determination of the starting time of an activity**
for an activity A_i which is not scheduled and whose predecessors in the graph $G(\alpha) = (V, E \cup \alpha)$ have all been scheduled, we set:
$u_i = max\{S_j + p_j | (A_j, A_i) \in E\}$.
t is the first time after u_i when there are sufficient resources for activity A_i;
if no such u_i exists, go to (5).
3) **Assignment of the activity**
$S_i = t$;
if there are activities which are not scheduled, go to (2);
otherwise, go to (4).
4) **Writing of the solution**
return schedule $S = \{S_i | A_i \in V\}$.
5) **Solution non-existent**
return (no feasible schedule).

Figure 10.2. *The strict order algorithm.*

1. Note an arbitrage actually defines a strict total order on the set of activities (see section 1.5)

The strict order algorithm can be adapted to a graph $G = (V, E, p)$ which contains negative cycles [CAR 84], but it does not always converge. A negative arc from A_i to A_j with a negative value v_{ij} implies that activity A_i has to start before time $S_j + |v_{ij}|$. A negative cycle is a cycle with a negative value.

We can apply the strict order algorithm to Example 10.1 with an arbitrage $\alpha = (A_1, A_2, A_3, A_4)$. We obtain the following schedule: $S = \{S_1 = 0, S_2 = 0, S_3 = 3, S_4 = 3, S_5 = 6\}$.

Without loss of generality we assume here that we always have only one resource, that the availability of this resource is B, and that a_i defines the number of units of resource required for activity A_i.

We can also associate an AND/OR precedence graph with an arbitrage by using forbidden sets. A forbidden set F (see also section 1.6) is a set of jobs without any precedence relations among them but that cannot be scheduled in parallel as their consumption of some resource exceeds the availability of that resource. A forbidden set is minimal, if every proper subset can be scheduled in parallel. Let $F = \{f_1, f_2..., f_s\}$ be a minimal forbidden set where activities are sorted by increasing order in the arbitrage, and let us introduce OR constraints between activities $f_1, ..., f_{s-1}$ and f_s. For an arbitrage $\alpha = (i_1, i_2, ...i_n)$ we check every minimal forbidden set of activities and then transform the problem into a project scheduling problem with AND/OR constraints. In Example 10.1 with the arbitrage $\alpha = (A_1, A_2, A_3, A_4)$ we have three minimal forbidden sets: $\{A_1, A_2, A_3\}$, $\{A_1, A_3, A_4\}$ and $\{A_2, A_3, A_4\}$. We therefore introduce OR precedence constraints between $\{A_1, A_2\}$ and A_3, $\{A_1, A_3\}$ and A_4, $\{A_2, A_3\}$ and A_4. The associated AND/OR precedence graph for Example 10.1 with an arbitrage $\alpha = (A_1, A_2, A_3, A_4)$ is shown in Figure 10.3.

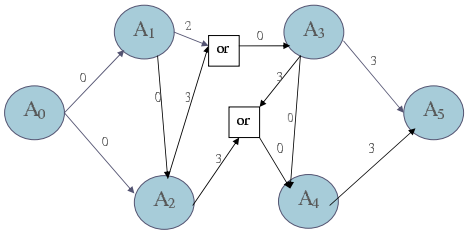


Figure 10.3. *The associated AND/OR precedence graph for Example 10.1.*

In fact, though there may be at most 2^n forbidden sets for a project where n is the number of activities, many of them can be eliminated by reductions. In Example 10.1, all two of these OR precedence constraints can be eliminated by reductions. A_1 precedes A_3, so we can ignore $\{A_1, A_2, A_3\}$. And for $\{A_1, A_3, A_4\}$, as A_1 is a predecessor of A_3, we can remove A_3 and take A_1 as a direct predecessor of A_4. In Figure 1.4, we show the simplified associated graph.

We can apply the algorithm proposed by Möhring *et al.* [MÖH 04] to the graph in Figure 10.4 and obtain the following: $S = \{S_1 = 0, S_2 = 0, S_3 = 3, S_4 = 3, S_5 = 6\}$. The result is identical to that obtained using the strict order algorithm.

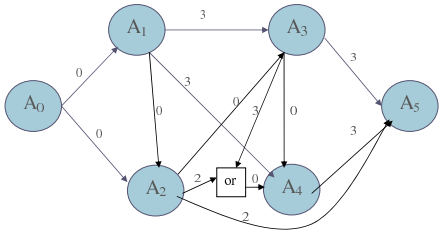


Figure 10.4. *The simplified associated AND/OR precedence graph for Example 10.1.*

10.4. Scheduling problems with production and consumption of a non-renewable resource

In 2006, Carlier *et al.* [CAR 06] developed a model which also allows resource production by tasks, and proposed several arbitrage methods to solve the problem.

We consider a set $V = \{A_0, A_1, ..., A_n, A_{n+1}\}$ of activities to be scheduled where activity A_0 (respectively A_{n+1}) is a fictitious beginning (respectively termination) activity. The time lag from activities A_i to A_j equals to v_{ij}. If $v_{ij} > 0$, then activity A_j cannot start before time $S_i + |v_{ij}|$, where S_i is the starting time of activity A_i. If $v_{ij} < 0$, then activity A_i must start before time $S_j + |v_{ij}|$. A directed graph $G = (V, E)$ is assumed to have no directed cycles and is an AoN graph with nodes as activities and arcs as precedence constraints. Preemption is not allowed. We also consider resource constraints. Each activity A_i needs a fixed number a_i of units of a non-renewable resource at the start of its execution. If $a_i > 0$, then activity A_i consumes $|a_i|$ units of resource; conversely, if $a_i < 0$, then activity A_i produces $|a_i|$ units of resource.

To illustrate the presentation we use the following example.

EXAMPLE 10.2.– Let $V = \{A_0, A_1, A_2, A_3, A_4, A_5\}$ be the set of activities where activity A_0 (respectively A_5) is a fictitious beginning (respectively termination) activity. $a_0 = 0, a_1 = 3, a_2 = -2, a_3 = -2, a_4 = -3, v_{01} = 0, v_{02} = 0, v_{14} = 3, v_{23} = 2, v_{31} = -2, v_{35} = 2, v_{41} = -3$ and $v_{45} = 3$.

The graph resulting from Example 10.2 is shown in Figure 10.5. The arc labels represent the time lags (v_{ij}) and the node labels represent the resource demands (a_i) of the activities. The label on node A_0 is the number of units of resource available to the project at the outset.

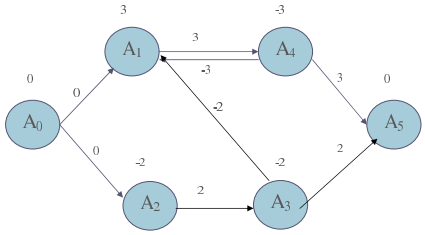


Figure 10.5. *The graph resulting from Example 10.2.*

This model successfully represents activities which require renewable resources or which consume non-renewable resources. An activity with processing time x and requiring y ($y > 0$) units of a renewable resource can be represented in this model by activities A_i and A_j, where activity A_i consumes y units of resource and activity A_j produces y units of resource, and by two arcs (A_i, A_j) and (A_j, A_i), where $v_{ij} = x$ and $v_{ji} = -x$. In Example 10.2 we can view A_1 and A_4 as activities with processing times $p = 3$ and requiring 3 units of the renewable resource. For activities with consumption or production of non-renewable resources, we apply this model directly, with $a_i > 0$ representing consumption and $a_i < 0$ representing production. Finding a feasible schedule for such a model is NP-hard, since RCPSP with time windows alone is NP-hard [JOH 05].

An algorithm to calculate the earliest starting time schedule associated with an arbitrage is shown below.

A complete arbitrage α is compatible if there exists a feasible schedule $S = \{S_1, S_2, ..., S_n\}$ such that for each arc (A_i, A_j) in α we have $S_j \geq S_i$. Without loss of generality, we can assume that arbitrage $\alpha = (A_1, A_2..., A_n)$.

The sum of productions should be larger than the sum of consumptions. Thus, in order to have a feasible schedule, it is necessary to have

$$\sum_{j=0}^{n} a_j \leq 0.$$

In order to check the compatibility of a complete arbitrage, we need to check the existence of a feasible schedule which satisfies resource constraints in the graph $G(\alpha) = (V, E \cup \alpha)$. The idea is to transform the problem into a precedence constraints problem without resource constraints. To this end we introduce the set of implicit precedence constraints β below.

The only resource constraint in our problem is that we can start an activity when there are sufficient resources for it. Now assume that we want to start activity A_r and

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

In this case, if there is no other activity which produces resources starting at the same time, the resource constraint will not be satisfied. So we have to force a production activity A_t to start at the same time as activity A_r, where

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

For $A_s, A_t \in \alpha$ where $s < t$, we say that the arc from A_t to A_{s+1} of value 0 is *implied* by α if and only if for every r where $s < r < t$ the following conditions are satisfied:

$$\sum_{j=0}^{s} a_j \leq 0 \tag{10.1}$$

$$\sum_{j=0}^{r} a_j > 0, \qquad \forall r \in]s, t[\tag{10.2}$$

$$\sum_{j=0}^{t} a_j \leq 0. \tag{10.3}$$

$$\tag{10.4}$$

β is the set of all arcs implied by α in this way. In order to generate β, let

$$D = \{A_j | \sum_{k=0}^{j} a_k \leq 0\}.$$

We suppose that there are m elements in D and $D = \{A_{j_1}, A_{j_2}, ..., A_{j_m}\}$ such that $j_1 \leq j_2 \leq ... \leq j_m$. If $j_1 \neq 0$, we add the arc (A_{j_1}, A_0) of zero value to β, and for every s where $j_{s+1} > j_s + 1$ we add the arc $(A_{j_{s+1}}, A_{j_s+1})$ of zero value to β.

THEOREM 10.1.– *A complete arbitrage $\alpha = \{(A_1, A_2), (A_2, A_3), ..., (A_{n-1}, A_n)\}$ is compatible if and only if the directed graph $G(\alpha, \beta) = (V, E \cup \alpha \cup \beta)$ contains no directed cycles of strictly positive value, where β contains all implicit precedence constraints deduced from α.*

If the directed graph $G(\alpha, \beta) = (V, E \cup \alpha \cup \beta)$ contains no positive directed cycles, we can apply the algorithm of Floyd-Warshall [LAW 76] to obtain an optimal schedule. This schedule will satisfy all precedence and resource constraints, since a

graph without strictly positive directed cycles is a sufficient condition for the Floyd-Warshall algorithm to find a feasible schedule. The arbitrage is therefore compatible.

Let us now suppose that the complete arbitrage α is compatible and that there is a strictly positive cycle $(A_{v_1}, A_{v_2}, ..., A_{v_p}, A_{v_1})$ in the graph $G(\alpha, \beta)$. So $S_{v_1} > S_{v_1}$, which is a contradiction for the existence of a feasible schedule.

Input:
A complete arbitrage $\alpha = \{(A_{i_1}, A_{i_2}), (A_{i_2}, A_{i_3}), ..., (A_{i_{n-1}}, A_{i_n})\}$
A conjunctive graph $G = (V, E)$
Output:
An Earliest Start Time Schedule S for the arbitrage α.
(1) Generate β, which contains all implicit precedence constraints deduced from α;
(2) Use the Floyd-Warshall algorithm to compute S in graph $G(\alpha, \beta) = (V, E \cup \alpha \cup \beta)$.

Figure 10.6. *Algorithm to compute the earliest starting time schedule of a complete arbitrage.*

The algorithm yields the earliest starting time schedule if the given arbitrage is compatible. Its complexity is $O(n^3)$. It works even where values are negative.

10.5. Discussion

Using the complete arbitrage method for enumeration would be very costly. For this reason we have also proposed restricting the arbitrage to consumption activities [CAR 06]. We are now working on algorithms for computing earliest schedules in this restricted case. We can also restrict the arbitrage to production activities when computing latest schedules. The arbitrages may then be used to construct both exact and approximate methods.

Chapter 11

Activity Insertion Problem in a RCPSP with Minimum and Maximum Time Lags

11.1. Introduction

In the standard RCPSP, the precedence relations are simple: an activity cannot start before the end of all its predecessors. In other words, between the start time of an activity A_i and the start time of its successor A_j there is a minimal distance (or minimum time lag) equal to the duration of A_i. The RCPSP with minimum and maximum time lags (RCPSP/max) involves generalized precedence relations where the minimum time lag between A_i and A_j can be any non-negative value and where there is possibly a maximum allowed time lag between the start time of A_i and the start time of A_j.

Independently of makespan minimization, simply finding a resource-feasible schedule that respects both the minimum and maximum time lags is NP-hard [BAR 88] whereas, for the standard RCPSP, this problem is polynomial. For that reason, the RCPSP/max problem has been extensively studied in the scheduling literature. For a thorough analysis of this problem, we recommend the book of Neumann, Schwindt and Zimmermann [NEU 03]. In this chapter, unlike most approaches that focused on finding a global schedule that minimizes the project duration, we illustrate the difficulties brought by the maximum time lags by considering the apparently simple problem of inserting a single activity inside an existing schedule.

Chapter written by Christian ARTIGUES, Cyril BRIAND.

That problem aims at finding an insertion position that is feasible with regards to the min/max time lags and resource constraints and that preserves the partial schedule structure. The objective is to minimize the project duration increase. Such a problem arises in local search procedures for reinsertion neighborhoods that unschedule an activity and insert it at another position and also for reactive insertion of unexpected activities. The interest of preserving the structure of the partial schedule is twofold. First, it makes it possible to decrease the size of the search space, which is necessary for both neighborhood search and reactive scheduling. Second, this policy tends to minimize the disturbances during the online project execution phase, which is referred to as ensuring the schedule stability. We define the structure of the schedule by means of resource flow networks, inducing additional precedence constraints. The problem has already been addressed for the standard RCPSP in [ART 00, ART 03] where an O(n^2q) optimal solving procedure has been proposed where n is the number of activities and q is the number of resources. Unfortunately, this procedure is unable to tackle minimum and maximum time lags . In [GRÖ 07], insertion problems in a general disjunctive scheduling framework capturing a variety of job shop scheduling problems and insertion types are considered and lower and upper bound procedures are developed. Since disjunctive scheduling is a particular case of the RCPSP where all resources have unit availability, the proposed procedures do not apply to the problem considered in this paper. We show that the considered insertion problem is NP-hard for the RCPSP with minimum and maximum time lags. When maximum time lags are ignored, a $O(n^2q)$ algorithm is provided by a direct extension of the one proposed in [ART 03].

11.2. Problem statement

As for the standard RCPSP (see Chapter 1), activities constituting the project are identified by a set $V = \{A_0, \ldots, A_{n+1}\}$. Activity A_0 represents by convention the start of the schedule and activity A_{n+1} represents symmetrically the end of the schedule. The set of non-dummy activities is identified by $\mathcal{A} = \{A_1, \ldots, A_n\}$. p_i denotes the duration of activity i with $p_0 = p_{n+1} = 0$. We assume in this chapter that $p_i > 0, \forall A_i \in \mathcal{A}$.

A valued activity-on-node graph $G(V, E, l)$ is defined where nodes correspond to activities and arcs correspond to precedence relations. Each arc $(A_i, A_j) \in E$ is valued by an integer time lag l_{ij}. $l_{ij} \geq 0$ corresponds to a minimum time lag of l_{ij} units stating that A_j has to start at least l_{ij} time units after the start time of A_i. In the standard RCPSP, only minimum time lags are considered and $l_{ij} = p_i$ for each arc $(A_i, A_j) \in E$. $l_{ij} \leq 0$ corresponds to a of $-l_{ij}$ units stating that A_i has to start at the latest l_{ij} time units after the start time of A_j. Resource constraints are defined as in the standard RCPSP. $\mathcal{R} = \{R_1, \ldots, R_q\}$ denotes the set of resources. B_k denotes the availability of R_k. b_{ik} represents the amount of resource R_k used during the execution of i.

We assume that we have at least the following arcs in E: an arc (A_0, A_i) per activity $A_i \in \mathcal{A}$ valuated by $l_{0i} \geq 0$ and an arc (A_i, A_{n+1}) per activity $A_i \in \mathcal{A}$ valuated by $l_{i(n+1)} = p_i$.

A solution S is a schedule giving the start time S_i of each activity A_i. S is feasible if it is compatible with the generalized precedence constraints (11.1) and the resource constraints (11.2) defined in Chapter 1 and recalled below where $\mathcal{A}_t = \{i \in \mathcal{A} \,|S_i \leq t < S_i + p_i\}$ represents the set of non-dummy activities in process at time t.

$$S_j - S_i \geq l_{ij} \quad \forall (A_i, A_j) \in E \tag{11.1}$$

$$\sum_{A_i \in \mathcal{A}_t} b_{ik} \leq B_k \quad \forall R_k \in \mathcal{R}, \forall t \geq 0 \tag{11.2}$$

The makespan of a schedule S is equal to S_{n+1}, the start time of the end activity. The above-defined set $\mathcal{A}_t$ and constraints state that an activity cannot be interrupted once it is started. This is referred to as not allowing *preemption*. The RCPSP with minimum and maximum time lags is the problem (P) of finding a non-preemptive schedule S of minimal makespan S_{n+1} such that $S_0 = 0$ and subject to precedence constraints 11.1 and resource constraints 11.2. The problem does not necessarily have a solution. This can be due to an inconsistency of the precedence constraints (11.1), independently of the resource constraints. To verify the consistency of the precedence constraints E, a distance matrix $(\delta_{ij})_{i,j \in V^2}$ can be computed where δ_{ij} is the length of the longest path between i and j in $G(V, E, l)$. The problem has a solution if and only if the precedence constraints are consistent. (δ_{ij}) can be computed in $\mathcal{O}(n^3)$ by the Floyd-Warshall algorithm. More precisely, there is no solution if there is a cycle of positive length in $G(V, E, l)$. In the case the temporal constraints are consistent, the problem does not necessarily have a solution neither and the corresponding decision problem is NP-hard [BAR 88].

We use the concept of resource-flow network (introduced in Chapter 1) to represent the solutions to (P) [FOR 97, ART 00, ART 03, LEU 04, NEU 03]. A resource flow f is a $(n+2) \times (n+2) \times m$ matrix verifying equations (11.3-11.5) defined in Chapter 1 and recalled below:

$$f_{ij}^k \geq 0 \quad \forall A_i \in \mathcal{A} \cup \{A_0\}, \forall A_j \in \mathcal{A} \cup \{A_{n+1}\}, \forall R_k \in \mathcal{R} \tag{11.3}$$

$$\sum_{A_i \in \mathcal{A} \cup \{A_{n+1}\}} f_{0i}^k = \sum_{A_i \in \mathcal{A} \cup \{A_0\}} f_{i(n+1)}^k = B_k \quad \forall R_k \in \mathcal{R} \tag{11.4}$$

$$\sum_{A_j \in \mathcal{A} \cup \{A_{n+1}\}} f_{ij}^k = \sum_{A_j \in \mathcal{A} \cup \{A_0\}} f_{ji}^k = b_{ik} \quad \forall A_i \in \mathcal{A}, \forall R_k \in \mathcal{R} \tag{11.5}$$

f_{ij}^k denotes the number of resource R_k units transferred from activity A_i to activity A_j. We define $E(f)$ as the set of arcs induced by a resource flow[1] f, i.e. $E(f) = \{(A_i, A_j) \in V^2 | \exists k \in \mathcal{R}, f_{ij}^k > 0\}$. $E(f)$ can be seen as the set of precedence constraints represented by the flow. Consider graph $G^f = G(V, E \cup E(f), l(f))$ where $l(f)$ is defined as follows. Each arc $(i,j) \in E$ with $(i,j) \notin E(f)$ is valuated by l_{ij}. Each arc $(i,j) \in E(f)$ with $(i,j) \notin E$ is valuated by p_i. Each arc $(i,j) \in E \cap E(f)$ is valuated by $\max(p_i, l_{ij})$.

Let $(\delta_{ij}^f)_{A_i, A_j \in V}$ denotes the distance matrix associated with flow f. δ_{ij}^f is the length of the longest path from A_i to A_j in G^f. The precedence constraints induced by the flow are consistent if and only if there is no cycle of positive length in G^f. In particular, $\delta_{0i}^f, \forall A_i \in V$ denotes the earliest start schedule associated with the flow f. We set $\delta_{ij}^f = -\infty$ if there is no path from A_i to A_j in G^f.

PROPOSITION 11.1.– *(P) can be defined as the problem of finding a resource flow f verifying (11.3-11.5), such that G^f does not admit a cyle of positive length and such that $\delta_{0(n+1)}^f$ is minimal.*

Proof. The arguments used in Lemmas 1.1 and 1.2 (Chapter 1) and issued from [LEU 04] for the standard RCPSP can be easily transposed for the considered problem. Similarly, the ones used in [NEU 03] (section 2.13) for the RCPSP with minimum/maximum time lags and sequence-dependent setup times hold. ■

We consider a partial solution in which all activities but one, denoted by activity A_x ($0 < x < n+1$), have been scheduled. This amounts to considering a complete solution to problem (P_x) identical to (P) except that $b_{xk} = 0, \forall k \in \mathcal{R}$.

Consider an example comprising five real activities and a single resource of three units issued from ([NEU 03], Ch. 2, P. 26). Durations and resource demands are given in Table 11.1.

Minimum and maximum time lags and the corresponding the project network are displayed on the left side of Figure 11.1.

We consider a partial schedule for (P) in which activity A_1 is not scheduled, corresponding to a complete solution of (P_1), displayed in Figure 11.2. The resource flow

1. In Chapter 1, a strict order $P(f)$ is defined as the transitive closure of $E(f)$

A_i	0	1	2	3	4	5	6
p_i	0	6	4	2	4	2	0
b_i	0	1	2	2	2	3	0

Table 11.1. *Durations and resource demands*

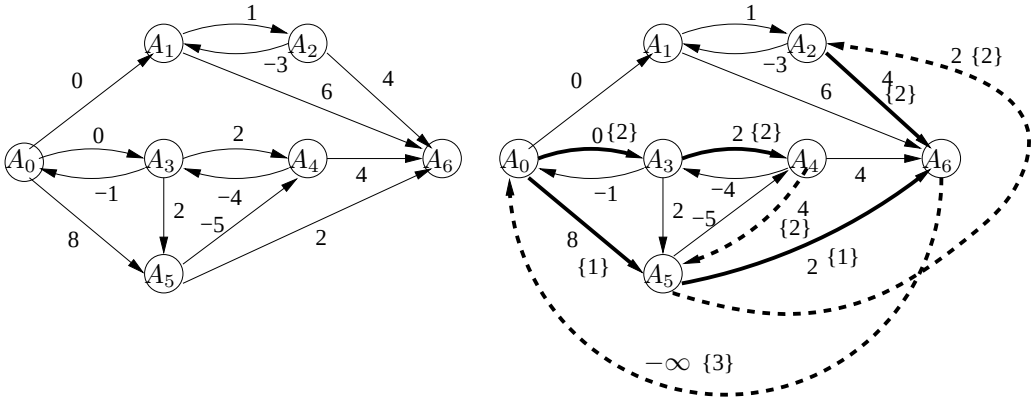


Figure 11.1. *Minimum/maximum time lags (left side) and solution of (P_1) (right side)*

corresponding to this schedule is displayed on the right side of Figure 11.1. Thin arcs correspond to the original precedence constraints only while thick arcs are induced by the flow (the flow value is displayed between braces). Plain thick arcs belong to $E \cup E(f)$ while thick dotted arcs are only induced by the flow.

The problem considered in this chapter amounts to computing a solution to (P) by inserting activity A_x in the flow f associated with the solution of (P_x) in such a way that the resource flow assigned to A_x can only be taken from f. More formally we define the considered insertion problem (IP_{xf}) as the problem to find q_{ij}^k such that $0 \leq q_{ij}^k \leq f_{ij}^k$ for all $i, j \in V$, $\forall k \in \mathcal{R}$ and such that the flow f' defined by:

$$f'^k_{xj} = \sum_{A_i \in V} q^k_{ij} \qquad \forall A_j \in V, \forall R_k \in \mathcal{R} \tag{11.6}$$

$$f'^k_{ix} = \sum_{A_j \in V} q^k_{ij} \qquad \forall A_i \in V, \forall R_k \in \mathcal{R} \tag{11.7}$$

$$f'^k_{ij} = f^k_{ij} - q^k_{ij} \qquad \forall A_i, A_j \in V, \forall R_k \in \mathcal{R} \tag{11.8}$$

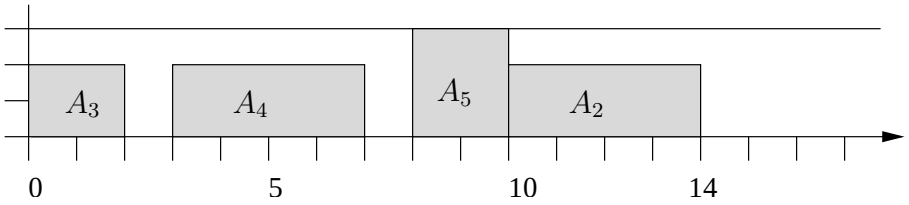


Figure 11.2. *Partial solution*

verifies equations (11.3-11.5), does not create a cycle of positive length in $G^{f'}$ and minimizes $\delta^{f'}_{0(n+1)}$. Note we have in any case $\delta^{f'}_{0(n+1)} \geq \delta^{f}_{0(n+1)}$ since by transitivity, any precedence constraint induced by f is also induced by f'.

11.3. Insertion positions

We define an insertion position (see also section 6.2.2) as an ordered pair of activity sets (α, β) such that $(\alpha, \beta) \in V^2$, $\alpha \cap \beta = \emptyset$ and verifying:

$$\sum_{A_i \in \alpha, A_j \in \beta} f^k_{ij} \geq b_{xk}, \forall k \in \mathcal{R}. \tag{11.9}$$

An insertion position induces a set of arcs $E(\alpha, \beta) = \{(A_a, A_x)\}_{A_a \in \alpha} \cup \{(A_x, A_b)\}_{A_b \in \beta}$. We consider graph $G^{f\alpha\beta} = G(V, E \cup E(f) \cup E(\alpha, \beta), l(f, \alpha, \beta))$ where $l(f, \alpha, \beta)$ denotes the arc valuation induced by (α, β). $l(f, \alpha, \beta)$ defines the same values as $l(f)$ except for the following arcs. For each $A_a \in \alpha$, (A_a, A_x) is valued by p_a if $(A_a, A_x) \notin E$ or by $\max(p_a, l_{ax})$ otherwise. For each $A_b \in \beta$, (A_x, A_b) is valued by p_x if $(A_x, A_b) \notin E$ or by $\max(p_x, l_{xb})$ otherwise. Let $\delta^{f\alpha\beta}_{ij}$ denotes the length of the longest path from A_i to A_j in $G^{f\alpha\beta}$.

In what follows, we refer to a feasible insertion position for (IP$_{xf}$) as an insertion position (α, β) such that there is no positive length cycle in $G^{f\alpha\beta}$. Obviously, solving (IP$_{xf}$) amounts to finding a feasible insertion position (α, β) such that $\delta^{f\alpha\beta}_{0(n+1)}$ is minimized. In other words, if (α, β) is a feasible insertion position for (IP$_{xf}$), $\delta^{f\alpha\beta}_{0(n+1)}$ is actually an upper bound of the makespan. The bound is tight for the insertion position of minimal makespan.

11.4. Feasibility conditions under a makespan upper bound

The search variant of (IP$_{xf}$) is considered in this section. It is denoted by (FIP$_{xfv}$). The problem is to find a feasible solution to (IP$_{xf}$) with a makespan not greater than v.

This can be achieved by searching for a feasible solution to (IP$_{xf}$) after adding an arc from A_{n+1} to A_0 valued by $\max(-v, l_{(n+1)0})$. Let $E(f,v) = E(f) \cup \{(A_{n+1}, A_0)\}$ and $l(f,v)$ denote the so modified set of arcs and values. For a given insertion position (α, β), let $l(f,v,\alpha,\beta)$ the corresponding arc values. With these definitions, (FIP$_{xfv}$) amounts to searching for an insertion position (α, β) such that there is no positive length cycle in $G^{fv\alpha\beta} = G(V, E \cup E(f,v) \cup E(\alpha,\beta), l(f,v,\alpha,\beta))$.

As a preliminary remark, if there is a positive length cycle in G^{fv}, there is obviously no feasible solution to (FIP$_{xfv}$). Again, this can be determined in $O(n^3)$ by the Floyd-Warshall algorithm. However, according to the last remark of section 11.2, if f represents a feasible solution to (P$_x$), there is a positive length cycle in G^{fv} if and only if $v < \delta^f_{0(n+1)}$. In what follows, we assume $v \geq \delta^f_{0(n+1)}$ and we consider $(\delta^{fv}_{ij})_{A_i,A_j \in V}$ defined as the distance matrix corresponding to the all-pairs longest path lengths in G^{fv}.

The remaining of the section is devoted to the evaluation of the feasibility of an insertion position (α, β) (without generating graph $G^{fv\alpha\beta}$ nor computing matrix $(\delta^{f\alpha\beta}_{ij})$). For this purpose, we define the following three categories of elementary cycles in $G^{fv\alpha\beta}$:

• An elementary cycle of type $\mathcal{C}_1(v,\alpha,\beta)$ involves an arc (A_a, A_x) with $A_a \in \alpha$ and $l(f,v,\alpha,\beta)_{ax} = p_a$ and no arc (A_x, A_b) with $A_b \in \beta$. Note that in $G^{fv\alpha\beta}$, a longest path from A_x to A_a not traversing any node of β is also a longest path from A_x to A_a in G^{fv} of length δ^{fv}_{xa} (since the path cannot traverse A_x again). Therefore $L_1(v,\alpha,\beta) = \max_{A_a \in \alpha}(p_a + \delta^{fv}_{xa})$ is the length of the cycles of $\mathcal{C}_1(v,\alpha,\beta)$ having the largest length.

• An elementary cycle of type $\mathcal{C}_2(v,\alpha,\beta)$ includes an arc (A_x, A_b) with $A_b \in \beta$ and $l(f,v,\alpha,\beta)_{xb} = p_x$ and no arc (A_a, A_x) with $A_a \in \alpha$. Using the same arguments asthe ones use in the previous item, the length of the longest path from A_b to A_x in $G^{fv\alpha\beta}$ is equal to δ^{fv}_{bx}. Consequently, $L_2(v,\alpha,\beta) = p_x + \max_{A_b \in \beta}(\delta^{fv}_{bx})$ denotes the length of the cycle in $\mathcal{C}_2(v,\alpha,\beta)$ having the largest length.

• A cycle of type $\mathcal{C}_3(v,\alpha,\beta)$ involves two arcs $(A_a, A_x), (A_x, A_b)$ with $l(f,v,\alpha,\beta)_{ax} = p_a$ and $l(f,v,\alpha,\beta)_{xb} = p_x$. Again, the longest path in $G^{fv\alpha\beta}$ passes only through arcs existing in $G(V, E \cup E(f,v), l(v,f))$ and thus has length δ^{fv}_{ba}. Hence, let $L_3(v,\alpha,\beta) = p_x + \max_{(A_a,A_b)\in\alpha\times\beta}(p_a + \delta^{fv}_{ba})$ be the length of the cycle of $\mathcal{C}_3(v,\alpha,\beta)$ having the largest length.

In what follows, $\mathcal{C}_q(v,\alpha,\beta)$ and $L_q(v,\alpha,\beta)$ ($q = 1,2,3$) will be simply denoted $\mathcal{C}_q$ and L_q when there is no ambiguity. Figure 11.3 illustrates the three considered categories of cycles.

THEOREM 11.1.– *An insertion position (α, β) is feasible for (FIP$_{xfv}$) if and only if* $\max(L_1, L_2, L_3) \leq 0$.

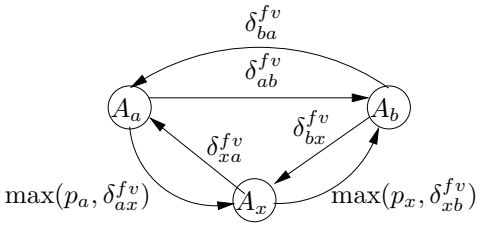


Figure 11.3. *Cycle analysis after an activity insertion*

Proof. This amounts to showing that there is no cycle of positive length in $G^{fv\alpha\beta}$ if and only if $\max(L_1, L_2, L_3) \leq 0$.

($\Rightarrow$ part) If there is no positive length cycle in $G^{fv\alpha\beta}$, since each L_i is not larger than an actual cycle length, then L_1, L_2 and L_3 are non positive.

($\Leftarrow$ part) We know that in a valued digraph there is no positive length cycle if and only if there is no elementary positive length cycle. Thus, it is sufficient to show that if $\max(L_1, L_2, L_3) \leq 0$, there is no elementary positive length cycle in $G^{fv\alpha\beta}$. Among the possible elementary cycles, we may restrict to the ones traversing at least one arc (A_a, A_x) with $A_a \in \alpha$ and $l(f, v, \alpha, \beta)_{ax} = p_a$ (type (i) arc) or one arc (A_x, A_b) with $A_b \in \beta$ and $l(f, v, \alpha, \beta)_{xb} = p_x$ (type (ii) arc). Indeed, any cycle traversing none of these arcs has a length equal to the length of a cycle in G^{fv} which is by hypothesis non-negative. Furthermore, at most one type (i) arc and at most one type (ii) arc may be traversed by any elementary cycle.

Suppose now that $\mathcal{C}$ denotes an elementary cycle of $G^{fv\alpha\beta}$ with a largest length but not belonging any category among $\mathcal{C}_1$, $\mathcal{C}_2$, nor $\mathcal{C}_3$. According to the presence of type (i) and (ii) arcs there are three possibilities. If $\mathcal{C}$ traverses exactly one type (i) arc and no type (ii) arcs, the path in $\mathcal{C}$ from A_x to A_a is not larger than δ_{xa}^{fv}. Indeed, by construction, for any activities A_i, A_j, A_k we have $\delta_{ij}^{fv} \geq \delta_{ik}^{fv} + \delta_{kj}^{fv}$. Consequently, there exists in $\mathcal{C}_1$ a cycle not shorter than $\mathcal{C}$. For the same reason, if $\mathcal{C}$ traverses exactly one type (ii) arc and no type (i) arcs, the path in $\mathcal{C}$ from A_b to A_x is not larger than δ_{bx}^{fv} and there exists in $\mathcal{C}_2$ a cycle not shorter than $\mathcal{C}$. Last, in the case $\mathcal{C}$ traverses exactly one type (i) arc and one type (ii) arc, the path in $\mathcal{C}$ from A_b to A_a is not larger than δ_{ba}^{fv} and there exists in $\mathcal{C}_3$ a cycle not shorter than $\mathcal{C}$. It follows that there exists an elementary longest length cycle that belongs either to $\mathcal{C}_1$, $\mathcal{C}_2$ or $\mathcal{C}_3$. ■

11.5. Computational complexity of the insertion problem with minimum and maximum time lags

In this section we prove that (IP$_{xf}$) is NP-hard. We show that the decision problem (DIP$_{xfv}$), stating whether there exists a feasible solution to (FIP$_{xfv}$) or not, is NP-complete by reduction from the independent set problem. We recall that, given a graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of edges, and an integer k, the independent set problem lies in finding a subset of vertices $\mathcal{W} \subseteq \mathcal{V}$ of cardinality k such that $\forall x, y \in \mathcal{W}$, $(x, y) \notin \mathcal{E}$ (i.e. the vertices of $\mathcal{W}$ are pairwise independent). Such a problem is known to be NP-hard. Let p denotes the number of vertices: $p = |\mathcal{V}|$.

In what follows, we show how to associate to any independent set problem an instance of the insertion problem with $n = 2p + 1$ non-dummy activities and a single resource of availability $B_1 = p$ (the resource index can be omitted).

The non-dummy activities are partitioned into three subsets $\Theta = \{A_1, \ldots, A_p\}$, $\Omega = \{A_{p+1}, \ldots, A_{2p}\}$ and $A_n = A_{2p+1}$, A_n being the activity to insert. All activities but A_n are such that $b_i = p_i = 1, \forall A_i \in \Theta \cup \Omega$. For A_n we have $p_n = 3$ and we set b_n to k. The resource flow f is such that

$$\begin{aligned} f_{0i} &= 1 \qquad \forall A_i \in \Theta \\ f_{i(p+i)} &= 1 \qquad \forall A_i \in \Theta \\ f_{j(n+1)} &= 1 \qquad \forall A_j \in \Omega. \end{aligned}$$

All other resource flows are zero. The precedence constraints E are such that there is an arc (A_0, A_i) with $l_{0i} = 0$, for each $A_i \in \Theta$. There is an arc (A_j, A_{n+1}) with $l_{j(n+1)} = 1$ for each $j \in \Omega$. There is an arc (A_i, A_{p+i}) with $l_{i(p+i)} = 2$ for each $A_i \in \Theta$. There is an arc (A_{n+1}, A_0) such that $l_{(n+1)0} = -5$ (i.e. the makespan has to be not greater than 5). For the inserted task A_n, there is an arc (A_0, A_n) and an arc (A_n, A_{n+1}) with $l_{0n} = 0$ and $l_{n(n+1)} = 3$, respectively.

The set of edges $\mathcal{E}$ of the graph of the independent set problem can be arbitrarily orientated. Let $G(\mathcal{V},\mathcal{U})$ denotes the graph so obtained. Now we create in E an arc (A_{p+j}, A_i) with $l_{(p+j)i} = -3$ for each $(i, j) \in \mathcal{U}$ to represent the edges in $\mathcal{E}$. Figure 11.4 shows how the insertion problem instance is built from an example graph of six nodes. Below each arc the time lag is indicated and the transferred resource flow is displayed between braces.

It can be easily checked that the minimal distance matrix δ^{fv} for the so-defined insertion problem instance (with $v = 5$) is such that for each pair A_i, A_j with $i \in \Theta$ and $j \in \Omega$, we have $\delta^{fv}_{ji} = -3$ if $(j - p, i) \in \mathcal{U}$ and $\delta^{fv}_{ji} = -4$ otherwise.

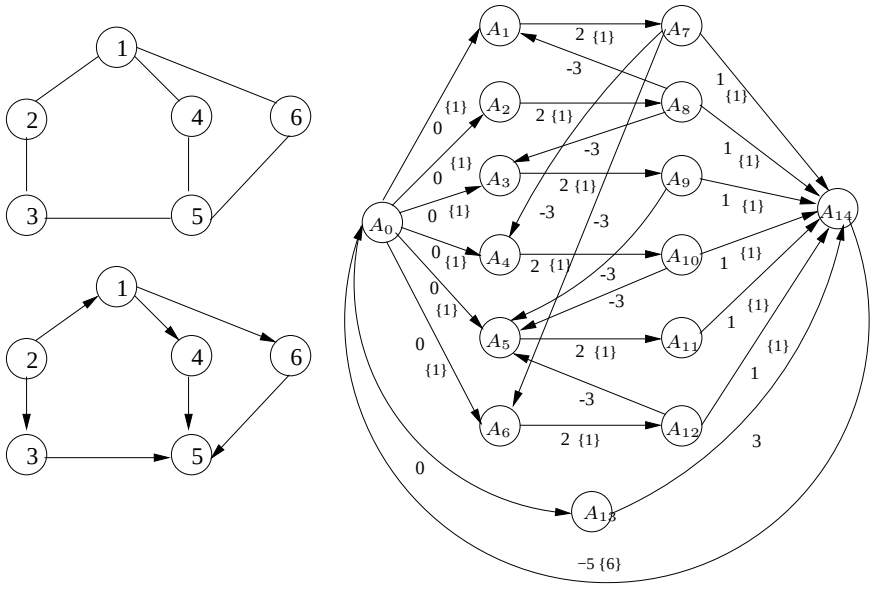


Figure 11.4. *Reduction from independent set*

Below we show that there exists an insertion position for A_n in G^{fv} if and only if there exists an independent set of cardinality k in $G(\mathcal{V}, \mathcal{E})$. First, one can observe that if W is an independent set of cardinality k, then there is a feasible insertion position (α, β). Indeed, if $\alpha = \{A_i | i \in W\}$ and $\beta = \{A_{p+i} | i \in W\}$ then, because the flow from $A_i \in \Theta$ to A_{i+p} is equal to 1, we have $\sum_{A_i \in \alpha, A_j \in \beta} f_{ij} = k$. Consequently (α, β) verifies condition (11.9). Moreover, there is no maximum time lags involving the inserted activity A_n and, consequently, no possible positive length cycles of types $\mathcal{C}_1$ nor $\mathcal{C}_2$. For any pair of nodes $x, y \in W$ we have no edge (x, y) in $\mathcal{E}$, therefore we have no arc $(x + p, y)$ in $\mathcal{U}$. As a result, $\forall A_j \in \Omega$, $\forall A_i \in \Theta$, we have $\delta_{ji}^{fv} = -4$. The largest cycle of type $\mathcal{C}_3$ has a length equal to $L_3 = \max_{(A_a, A_b) \in \alpha \times \beta}(p_a + p_x + \delta_{ba})$. Since $p_a = 1$ and $p_x = 3$, we have $L_3 = 0$. It follows that the insertion problem is feasible.

Now let us show that if the insertion is feasible, there is an independent set of cardinality k. Let (α, β) denotes the feasible insertion position. Obviously, the insertion position such that $\alpha = \{A_0\}$ and $\beta = \Theta \cup \Omega \cup A_{n+1}$ is not feasible since it will increase the project duration to 6, whereas it is bounded by 5. The same remark holds for the insertion position such that $\alpha = A_0 \cup \Theta \cup \Omega$ and $\beta = \{A_{n+1}\}$. Therefore, A_n can only be located between some activities of Θ and some activities of Ω (i.e. $\alpha \subset \Theta$ and $\beta \subset \Omega$). For a feasible (α, β) insertion position, since no positive length cycles have been generated, we have $\delta_{ba}^{fv} = -4 \, \forall A_a \in \alpha, \forall A_b \in \beta$. Subsequently there is no

arc $(b-p, a)$ in $\mathcal{U}$, $\forall A_a \in \alpha, \forall A_b \in \beta$ and no edge (x, y) in E, $\forall a \in \alpha$. Furthermore, since the insertion is feasible, we have $|\alpha| = k$ therefore α is an independent set of cardinality k. Figure 11.5 illustrates the correspondence between an independent set (bold nodes) and an insertion position.

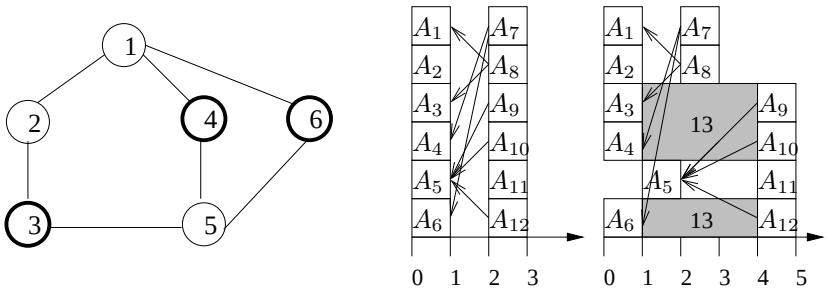


Figure 11.5. *A feasible independent set and its corresponding feasible insertion position*

While for the standard RCPSP, Artigues *et al.* proposed a polynomial algorithm to solve the insertion problem [ART 00, ART 03], the introduction of minimum and maximum time lags clearly makes the problem intractable despite of its apparent simplicity. This illustrates the deep modifications of the problem structure brought by the maximum time lags. To confirm this, we exhibit in the following section a polynomial algorithm to solve the insertion problem when only minimum time lags are considered.

11.6. A polynomial algorithm for the insertion problem with minimum time lags only

We suppose in this section that all time lags are non-negative. We still assume that f represents a feasible solution to (P_x) of makespan $\delta^f_{0(n+1)}$ corresponding to the length of the longest path between 0 and $n+1$ in G^f. We show that in this case the insertion problem (IP_{xf}) becomes polynomially solvable. First, let us prove that for any $v \geq \delta^f_{0(n+1)}$, the following lemma holds.

LEMMA 11.1.– $\forall A_i, A_j \in V^2$, $\delta^{fv}_{ij} = \max(\delta^f_{ij}, \delta^f_{0j} + \delta^f_{i(n+1)} - v)$.

Proof. Since δ^{fv}_{ij} is the length of the longest path in G^{fv}, let $\mathcal{P}$ denote such a longest path. If $\mathcal{P}$ traverses arc (A_{n+1}, A_0) (valued by $-v$ since we have no maximum time lags), by definition of the longest paths we have $\delta^{fv}_{ij} = \delta^{fv}_{0j} + \delta^{fv}_{i(n+1)} - v$. Since G^{fv} is obtained from G^f by adding arc (A_{n+1}, A_0), we have $\delta^{fv}_{0i} = \delta^f_{0i}$ and $\delta^{fv}_{j(n+1)} =$

$\delta^f_{j(n+1)}$. If $\mathcal{P}$ does not traverse arc (A_{n+1}, A_0) it has, for the same reason, length δ^f_{ij}. ■

With this property, we subdivide each cycle type $\mathcal{C}_q(v,\alpha,\beta)$ in $G^{fv\alpha\beta}$ into two subtypes, one being independent of v, as follows:

• $\mathcal{C}^1_1(v,\alpha,\beta)$ is the subset of elementary cycles $\mathcal{C}_1(v,\alpha,\beta)$ (traversing an arc (A_a, A_x) with $A_a \in \alpha$) such that $\delta^{fv}_{xa} = \delta^f_{0a} + \delta^f_{x(n+1)} - v$. Consequently, the longest cycle of this type has a length equal to $L^1_1(v,\alpha,\beta) = \max_{A_a\in\alpha}(\delta^f_{0a} + p_a) + \delta^f_{x(n+1)} - v$.

• $\mathcal{C}^2_1(\alpha,\beta)$ is the subset of elementary cycles $\mathcal{C}_1(v,\alpha,\beta)$ such that $\delta^{fv}_{xa} = \delta^f_{xa}$. The longest cycle of this type is of length $L^2_1(\alpha,\beta) = \max_{A_a\in\alpha}(\delta^f_{xa} + p_a)$. Note we have $L^2_1(\alpha,\beta) = -\infty$ if $\forall A_a \in \alpha$, $\delta^f_{xa} = -\infty$.

• $\mathcal{C}^1_2(v,\alpha,\beta)$ is the subset of elementary cycles $\mathcal{C}_2(v,\alpha,\beta)$ (traversing an arc (A_x, A_b) with $A_b \in \beta$) such that $\delta^{fv}_{bx} = \delta^f_{0x} + \delta^f_{b(n+1)} - v$. The longest cycle of this type is of length $L^1_2(v,\alpha,\beta) = \delta^f_{0x} + p_x + \max_{A_b\in\beta}(\delta^f_{b(n+1)}) - v$.

• $\mathcal{C}^2_2(\alpha,\beta)$ is the subset of elementary cycles $\mathcal{C}_2(v,\alpha,\beta)$ such that $\delta^{fv}_{bx} = \delta^f_{bx}$. The longest cycle of this type is of length $L^2_2(\alpha,\beta) = \max_{A_b\in\beta}(\delta^f_{bx}) + p_x$. We have $L^2_2(\alpha,\beta) = -\infty$ if $\forall A_b \in \beta$, $\delta^f_{bx} = -\infty$.

• $\mathcal{C}^1_3(v,\alpha,\beta)$ is the subset of elementary cycles $\mathcal{C}_3(v,\alpha,\beta)$ (traversing an arc (A_a, A_x) with $A_a \in \alpha$ and an arc (A_x, A_b) with $A_b \in \beta$) such that $\delta^{fv}_{ba} = \delta^f_{0a} + \delta^f_{b(n+1)} - v$. The longest cycle of this type is of length $L^1_3(v,\alpha,\beta) = \max_{A_a\in\alpha}(\delta^f_{0a} + p_a) + p_x + \max_{A_b\in\beta}(\delta^f_{b(n+1)}) - v$.

• $\mathcal{C}^2_3(\alpha,\beta)$ is the subset of elementary cycles $\mathcal{C}_3(v,\alpha,\beta)$ such that $\delta^{fv}_{ba} = \delta^f_{ba}$. The longest cycle of this type is of length $L^2_3(\alpha,\beta) = \max_{A_a\in\alpha, A_b\in\beta}(\delta^f_{ba} + p_a) + p_x$. We have $L^2_3(\alpha,\beta) = -\infty$ if $\delta^f_{ba} = -\infty$.

We consider now the insertion problem (IP$_{xf}$) relative to its search variant. An insertion position (α,β) is feasible for (IP$_{xf}$) if there exists an arbitrary large value $M \geq \delta^f_{0(n+1)}$ such that (FIP$_{xfM}$) is feasible. (IP$_{xf}$) amounts to finding the smallest $M \geq \delta^f_{0(n+1)}$ such that there exists a feasible insertion position for (FIP$_{xfM}$). Remarking that L^2_1, L^2_2 and L^2_3 do not depend on v, the following theorem can be stated:

THEOREM 11.2.– *An insertion position (α,β) is feasible for (IP$_{xf}$) if and only if it satisfies constraints (11.9) and* $\max\left(L^2_1(\alpha,\beta), L^2_2(\alpha,\beta), L^2_3(\alpha,\beta)\right) \leq 0$.

Proof. Searching a feasible solution to (IP$_{xf}$) amounts to searching a feasible solution to (FIP$_{xf\infty}$) where $v = \infty$. It follows that for any (α,β), $L^1_1(\alpha,\beta) = L^1_2(\alpha,\beta) =$

$L_3^1(\alpha,\beta) = -\infty$. It follows that there is no positive length cycle in $G^{f\alpha\beta}$ if and only if the considered property holds. ■

Thus, the feasibility of an insertion position only depends on the type 2 cycles.

THEOREM 11.3.– *If (α,β) is a feasible insertion position for (IP$_{xf}$), we have*

$$\delta_{0(n+1)}^{f\alpha\beta} = \max\left(\delta_{0(n+1)}^{f}, L_1^1(0,\alpha,\beta), L_2^1(0,\alpha,\beta), L_3^1(0,\alpha,\beta)\right). \tag{11.10}$$

Proof. Since (α,β) is feasible for (IP$_{xf}$), we have $L_1(v,\alpha,\beta) = L_1^1(v,\alpha,\beta)$, $L_2(v,\alpha,\beta) = L_2^1(v,\alpha,\beta)$ and $L_3(v,\alpha,\beta) = L_3^1(v,\alpha,\beta)$, $\forall v \geq \delta_{0(n+1)}^f$. The makespan of the solution represented by (α,β) is either equal to $\delta_{0(n+1)}^f$ or to the smallest value v such that $\max(L_1^1(v,\alpha,\beta), L_2^1(v,\alpha,\beta), L_3^1(v,\alpha,\beta)) \leq 0$ which yields the expression. ■

The makespan of a feasible insertion position depends on the type 1 cycles. Below, we derive a necessary and sufficient condition for the existence of a feasible insertion position and we define a feasible insertion position in the case the condition is verified.

Let $\gamma_0 = \{A_i \in \mathcal{A} | \delta_{ix}^f = 0 \text{ and } \delta_{xi}^f = 0\}$. The constraint $\delta_{ix}^f = 0$ and $\delta_{xi}^f = 0$ is called a synchronization constraint and γ_0 is the set of non-dummy activities that are synchronized with A_x. Note that A_{n+1} cannot be synchronized with A_x since $l_{x(n+1)} = p_x$. A_0 can only be synchronized with A_x if $l_{0x} = 0$. Note, that in the special case where $l_{0x} = l_{x0} = 0$, dummy activity A_0 does not belong to γ_0. Consequently, for any activity $A_i \in \gamma_0$ we cannot have $A_i \in \alpha$ (otherwise a cycle of length p_a would be issued) nor $A_i \in \beta$ (otherwise a cycle of length p_x would be issued). Furthermore, in any feasible schedule respecting the precedence constraints, the synchronized activities are constrained to start exactly at the same time. Consequently, there exists a solution to the insertion problem if the following inequalities hold.

$$\sum_{A_i \in \gamma_0} b_{ik} + b_{xk} \leq B_k \quad \forall R_k \in \mathcal{R} \tag{11.11}$$

Let (α_0,β_0) the insertion position such that $\alpha_0 = \{A_i \in V \setminus \gamma_0 | \delta_{ix}^f \geq 0\}$ and $\beta_0 = V \setminus (\gamma_0 \cup \alpha_0)$. It is easy to verify that (α_0,β_0) is a feasible insertion position if and only if γ_0 satisfies constraints (11.11). Moreover, if (α_0,β_0) is not feasible then there does not exist any feasible insertion position. Therefore, the existence of a feasible insertion position can be established and feasible insertion position (α_0,β_0) can be computed in $O(n|\mathcal{R}|)$. However, distance matrix δ_{ij}^f, needed as imput of these computations, is computed in $O(n^3)$. In the remaining of the section we present an algorithm to find an optimal insertion position.

Let (α, β) denotes a feasible insertion position for (IP$_{xf}$). Let μ denotes the subset of α such that $\mu = \{A_i \in \alpha | \delta_{0i} + p_i = \max_{A_a \in \alpha}(\delta_{0a} + p_a)\}$ and ν denotes the subset of β such that $\nu = \{A_i \in \beta | \delta^f_{i(n+1)} = \max_{A_b \in \beta}(\delta^f_{b(n+1)})\}$. Due to the expression of $\delta^{f\alpha\beta}_{0(n+1)}$ given by equation (11.10), there is no other feasible insertion position (α', β') such that $C_{\max}(\alpha', \beta') < C_{\max}(\alpha, \beta)$, $\mu \subseteq \alpha'$ and $\nu \subseteq \beta'$. With this property a branching scheme can be defined from the feasible insertion position (α_0, β_0) considering that any improving insertion position must verify either $\mu \not\subseteq \alpha'$ or $\nu \not\subseteq \beta'$, or both conditions.

Algorithm (7) is based on this property. At step 1, the set γ_0 is computed and step 2 checks whether the feasibility condition 11.11 is verified. If this is the case, the feasible insertion position (α_0, β_0) is generated at step 3. Starting with $\rho = 0$ a series of insertion positions $(\alpha_\rho, \beta_\rho)$ is generated from step 5 to step 17 until the flow sent from α_ρ to β_ρ becomes insufficient for the insertion of x. For each considered insertion position $(\alpha_\rho, \beta_\rho)$, a sub-series of included insertion positions (α, β_ρ) such that $\alpha \subset \alpha_\rho$ are evaluated in steps 7-13 starting with $\alpha = \alpha_\rho$ while there remains a sufficient amount of flow sent from α to β_ρ. At the end of loop 5-17, (α^*, β^*) is the best encountered insertion position, i.e. the one with the smallest $\delta^{f\alpha\beta}_{0(n+1)}$. Given an insertion position (α, β_ρ) with $\alpha \subseteq \alpha_\rho$, the next insertion sub-position is generated by removing from set α the set μ presented above (steps 11-12). Given an insertion position $(\alpha_\rho, \beta_\rho)$, the next insertion position $(\alpha_{\rho+1}, \beta_{\rho+1})$ is generated from step 14 to step 16, by considering set ν presented above and its subet $\nu' \subseteq \nu$ such that for any activity $A_i \in \nu'$, there is no path in G^f from A_x to A_i. Indeed, the activities belonging to ν but not to ν' can not precede A_x without causing the creation of a positive length cycle. Thus, $\beta_{\rho+1}$ is generated by removing ν from β_ρ and $\alpha_{\rho+1}$ is generated by inserting set ν' in α_ρ. The remaining steps 18-26 update flow f (actually inserting activity A_x) from the insertion position (α^*, β^*) following the principles presented in section 11.3.

If (IP$_{xf}$) has no solution, this is detected by the algorithm at step 2 by using the necessary and sufficient feasibility condition 11.11. Otherwise, we first show that OPTINSERT generates a finite series of feasible insertion positions. The finiteness of the algorithm can be easily proved. For each insertion position (α, β_ρ), there is a finite number of subpositions obtained by removing set μ at step 12; μ includes each time at least one activity and the flow set from $\alpha = \emptyset$ is zero so the stop condition of step 7 will be reached in less than n iterations starting from $\alpha = \alpha_\rho$. Also, set ν contains at least one activity, so the repeated removal of ν from β_ρ will reach the stop condition at least as soon as $\beta_\rho = \emptyset$ which can be obtained in less than n iterations from (α_0, β_0). Consequently there are most $O(n^2)$ generated insertion positions. Due to the flow computations, the algorithm can be implemented in $\mathcal{O}(n^2|\mathcal{R}|)$.

Algorithm 7 OPTINSERT(x, f) insert A_x into flow f minimizing the makespan

$\gamma_0 \leftarrow \{A_i \in \mathcal{A} | \delta_{ix}^f = 0 \text{ and } \delta_{xi}^f = 0\}$;
if $\sum_{A_i \in \gamma_0} b_{ik} + b_{xk} \leq B_k \quad \forall R_k \in \mathcal{R}$ **then**
  $\alpha_0 \leftarrow \{A_i \in V \setminus \gamma_0 | \delta_{ix}^f \geq 0\}$;$\beta_0 \leftarrow \{V \setminus (\gamma_0 \cup \alpha_0)\}$;
  $(\alpha^*, \beta^*) \leftarrow (\alpha_0, \beta_0)$; $\rho \leftarrow 0$;
  repeat
    $\alpha \leftarrow \alpha_\rho$;
    while $\sum_{(A_i, A_j) \in \alpha \times \beta_\rho} f_{ijk} \geq b_{xk} \quad \forall R_k \in \mathcal{R}$ **do**
      if $\delta_{0(n+1)}^{f\alpha\beta_\rho} < \delta_{0(n+1)}^{f\alpha^*\beta^*}$ **then**
        $(\alpha^*, \beta^*) \leftarrow (\alpha, \beta_\rho)$;
      end if
      $\mu \leftarrow \{A_i \in \alpha | \delta_{0i}^f + p_i = \max_{A_a \in \alpha}(\delta_{0a}^f + p_a)\}$;
      $\alpha \leftarrow \alpha \setminus \mu$;
    end while
    $\nu_\rho \leftarrow \{A_i \in \beta_\rho | \delta_{i(n+1)}^f = \max_{A_b \in \beta_\rho}(\delta_{b(n+1)}^f)\}$;
    $\nu'_\rho \leftarrow \{A_i \in \nu_\rho | \delta_{xi}^f = -\infty\}$;
    $\alpha_{\rho+1} \leftarrow \alpha_\rho \cup \nu'_\rho$; $\beta_{\rho+1} \leftarrow \beta_\rho \setminus \nu_\rho$; $\rho \leftarrow \rho + 1$;
  until $\exists R_k \in \mathcal{R}, \sum_{(A_a, A_b) \in \alpha_\rho \times \beta_\rho} f_{ijk} < b_{ik}$;
  for $R_k \in \mathcal{R}$ **do**
    $z_k \leftarrow b_{xk}$;
    for $A_i \in \alpha^*$ **do**
      for $A_j \in \beta^*$ **do**
        $\phi = \min(f_{ijk}, z_k)$; $z_k \leftarrow z_k - \phi$;
        $f_{ijk} \leftarrow f_{ijk} - \phi$, $f_{ixk} \leftarrow f_{ixk} + \phi$; $f_{xjk} \leftarrow f_{xjk} + \phi$;
      end for
    end for
  end for
else
  There is no solution to (IP$_x f$);
end if

THEOREM 11.4.– OPTINSERT *computes the optimal solution to (IP$_{xf}$) in* $O(n^2|\mathcal{R}|)$.

In order to illustrate how the previous algorithm works, let us consider a RCPSP example with 10 activities and two resources such that $B_1 = 7$ and $B_2 = 4$. The processing times and the resource demands are indicated on Table 11.2. Minimum time lags are displayed on Figure 11.6. We assume that all activities, except Activity 5 that has to be inserted, are already scheduled as shown on the Gantt diagram of Figure 11.8. The flow network f corresponding to this partial schedule is displayed on Figure 11.7 and gives a makespan $C_{\max} = 11$.

i	0	1	2	3	4	5	6	7	8	9	10	11
p_i	0	6	1	1	2	5	3	5	3	2	4	0
(b_{i1}, b_{i2})	(7,4)	(2,1)	(1,0)	(3,1)	(2,0)	(1,2)	(2,1)	(2,0)	(0,1)	(1,2)	(1,1)	(7,4)

Table 11.2. *Processing times and resource demands*

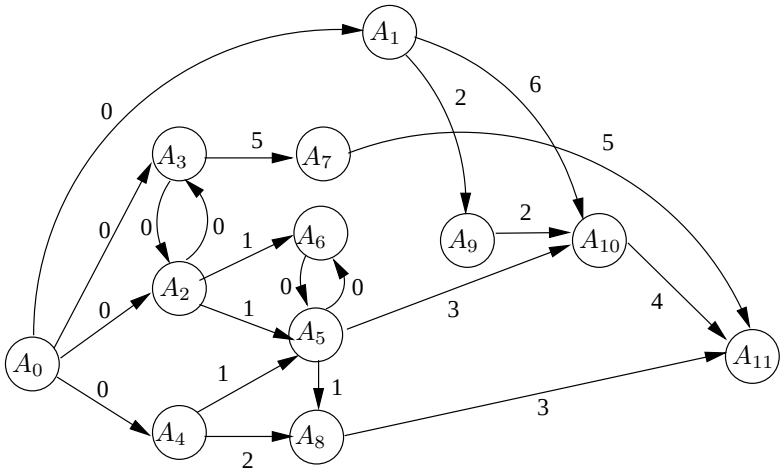


Figure 11.6. *Example of a RCPSP with minimum time lags*

The values of $\delta^f_{0i} + p_i$ and $\delta^f_{i(n+1)}$ can be computed using the Bellmann-Ford's algorithm. There are indicated on Table 11.3. We observe that the insertion problem is time-consistent since $\max(L^2_1(\alpha_0, \beta_0), L^2_2(\alpha_0, \beta_0), L^2_3(\alpha_0, \beta_0)) \leq 0$.

i	0	1	2	3	4	5	6	7	8	9	10	11
$\delta_{0i} + p_i$	0	7	1	1	2	4	4	10	7	5	11	11
$\delta_{i(n+1)}$	11	10	11	11	9	8	8	5	3	6	4	0

Table 11.3. *Earliest finishing and latest starting times*

Let us describe now the various insertion positions that Algorithm 7 considers for the insertion of A_5. At the very beginning ($\rho = 0$), the set γ_0, α_0 and β_0 are built up. $\gamma_0 = \{6\}$ since A_6 is the only activity which is synchronized with activity 5 ($S_5 =$

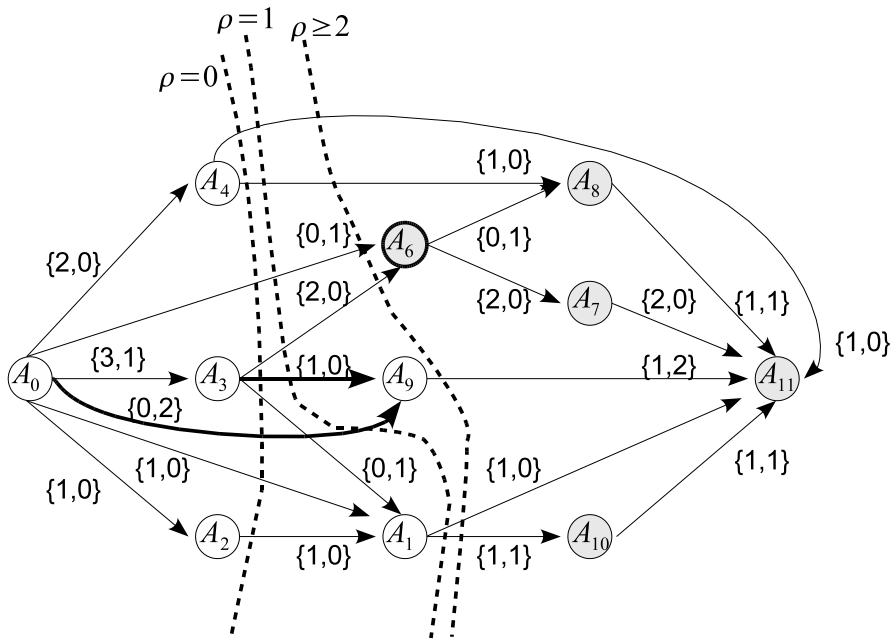


Figure 11.7. *Resource flow network*

S_6). Then we find $\alpha_0 = \{A_0, A_2, A_3, A_4\}$ and $\beta_0 = \{A_1, A_7, A_8, A_9, A_{10}, A_{11}\}$ (see the subcut $\rho = 0$ on Figure 11.7). This insertion position (α_0, β_0) is resource-feasible since, $\forall k$, $\sum_{(A_i,A_j)\in\alpha_0\times\beta_0} f_{ijk} \geq b_{5k}$. Table 11.4 displays the various insertion position that are explored assuming that β_0 is kept unchanged, as well as their evaluation. The insertion position (α_0, β_0) is feasible and leads to $\delta^{f\alpha\beta}_{0(n+1)} = \max\left(\delta^{f}_{0(n+1)}, L^1_1(0,\alpha_0,\beta_0), L^1_2(0,\alpha_0,\beta_0), L^1_3(0,\alpha_0,\beta_0)\right) = 15$. This value can be decreased to 14 by removing 4 and 2,3 from α. At the end of this phase, since only A_0 still belongs to α, the algorithm turns to consider the case $\rho = 1$, by transferring the activities belonging to $\nu'_0 = \nu_0 = \{A_1\}$ from β_0 to α_0.

Case $\rho = 1$ is described on Table 11.5. We know that $\alpha_1 = \{A_0, A_1, A_2, A_3, A_4\}$ and $\beta_1 = \{A_7, A_8, A_9, A_{10}, A_{11}\}$. For this case, three insertion positions can be considered. The best one (according to the greatest $L^1_i(0,\alpha,\beta)$ values) is found for $\alpha = \{A_0, A_2, A_3\}$. It allows for inserting A_5 in the partial schedule while keeping the $C_{\max}$ value unchanged. Note that at this point, since an insertion position that keeps the project duration unchanged has been found, it is not necessary any more to explore further insertion positions. Anyway, in order to explain all the features of the algorithm, we detail the other stages.

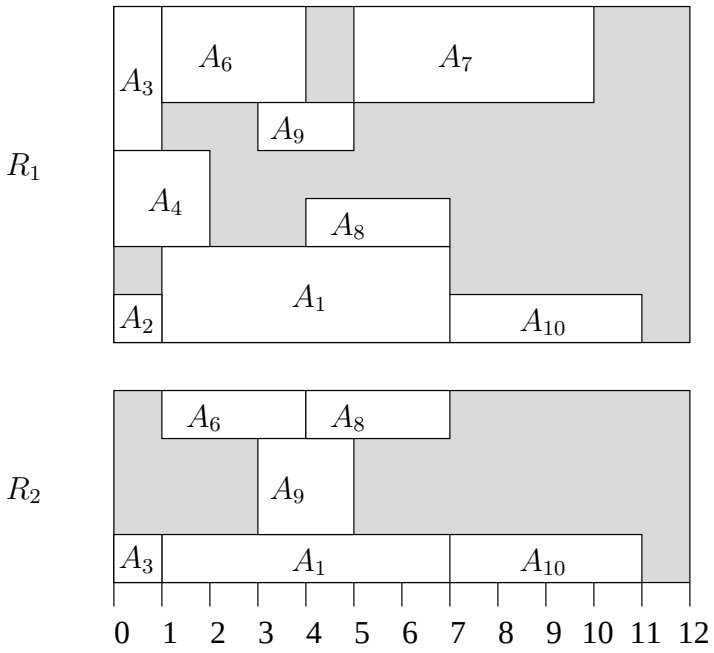


Figure 11.8. *Partial solution not including Activity* A_5

α	$\sum_{(A_i, A_j)\in\alpha\times\beta_\rho} f_{ijk}$	$L_1^1(0,\alpha,\beta_\rho)$	$L_2^1(0,\alpha,\beta_\rho)$	$L_3^1(0,\alpha,\beta_\rho)$	μ
$\{A_0,A_2,A_3,A_4\}$	(5,3)	10	14	15	$\{A_4\}$
$\{A_0,A_2,A_3\}$	(3,3)	9	14	14	$\{A_2,A_3\}$
$\{A_0\}$	(1,3)	8	14	13	Ø

Table 11.4. $\rho = 0$, $\alpha_0 = \{A_0, A_2, A_3, A_4\}$ *and* $\beta_0 = \{A_1, A_7, A_8, A_9, A_{10}, A_{11}\}$

Case $\rho = 2$ is considered next: the activities belonging to $\nu'_1 = \nu_1 = \{A_9\}$ from β_1 to α_1. It is described on Table 11.6. We have $\alpha_2 = \{A_0, A_1, A_2, A_3, A_4, A_9\}$ and $\beta_2 = \{A_7, A_8, A_{10}, A_{11}\}$. For this case, two insertion positions are considered but, as one can see on the table, they are both worse than the best known one. At this step, it is not any longer possible to transfer any activity from β to α because there is a path from A_5 to each activity A_7, A_8, A_{10} and A_{11} remaining in β ($\nu'_\rho = \emptyset\ \forall\rho \geq 2$). For this reason, these activities have been tinted gray on the resource

α	$\sum_{(A_i,A_j)\in\alpha\times\beta_\rho} f_{ijk}$	$L_1^1(0,\alpha,\beta_\rho)$	$L_2^1(0,\alpha,\beta_\rho)$	$L_3^1(0,\alpha,\beta_\rho)$	μ
$\{A_0,A_1,A_2,A_3,A_4\}$	(5,3)	15	10	16	$\{A_1\}$
$\{A_0,A_2,A_3,A_4\}$	(3,2)	10	10	11	$\{A_4\}$
$\{A_0,A_2,A_3\}$	(1,2)	9	10	10	$\{A_2,A_3\}$
$\{A_0\}$	(0,2)	-	-	-	-

Table 11.5. $\rho = 1$, $\alpha_1 = \{A_0, A_1, A_2, A_3, A_4\}$ *and* $\beta_1 = \{A_7, A_8, A_9, A_{10}, A_{11}\}$

network of Figure 11.7. Therefore, these activities can only be removed from β until the remaining flow becomes lower than the resource demand. According to the $\delta_{i(n+1)}$ values, A_7 is removed first (see Table 11.7), then A_{10} (see Table 11.8) and lastly A_8 (see Table 11.9). As one can see on Tables 11.7-11.9, no insertion positions are found that is better than the best one already known at this point (i.e. $\alpha^* = \{A_0, A_2, A_3\}$ and $\beta^* = \{A_7, A_8, A_9, A_{10}, A_{11}\}$). Figure 11.9 displays the obtained schedule after A_5 insertion. Note that only the resource flows coming from A_0 and A_3 are used (whereas A_2 also belongs to α^*) since $f_{2jk} = 0\ \forall j \in \beta^*$.

α	$\sum_{(A_i,A_j)\in\alpha\times\beta_\rho} f_{ijk}$	$L_1^1(0,\alpha,\beta_\rho)$	$L_2^1(0,\alpha,\beta_\rho)$	$L_3^1(0,\alpha,\beta_\rho)$	μ
$\{A_0,A_1,A_2,A_3,A_4,A_9\}$	(5,3)	15	9	15	$\{A_1\}$
$\{A_0,A_2,A_3,A_4,A_9\}$	(3,2)	13	9	13	$\{A_9\}$
$\{A_0,A_2,A_3,A_4\}$	(1,0)	-	-	-	-

Table 11.6. $\rho = 2$, $\alpha_2 = \{A_0, A_1, A_2, A_3, A_4, A_9\}$ *and* $\beta_2 = \{A_7, A_8, A_{10}, A_{11}\}$

α	$\sum_{(A_i,A_j)\in\alpha\times\beta_\rho} f_{ijk}$	$L_1^1(0,\alpha,\beta_\rho)$	$L_2^1(0,\alpha,\beta_\rho)$	$L_3^1(0,\alpha,\beta_\rho)$	μ
$\{A_0,A_1,A_2,A_3,A_4,A_9\}$	(5,3)	15	9	14	$\{A_1\}$
$\{A_0,A_2,A_3,A_4,A_9\}$	(3,2)	13	8	12	$\{A_9\}$
$\{A_0,A_2,A_3,A_4\}$	(1,0)	-	-	-	-

Table 11.7. $\rho = 3$, $\alpha_3 = \{A_0, A_1, A_2, A_3, A_4, A_9\}$ *and* $\beta_3 = \{A_8, A_{10}, A_{11}\}$

α	$\sum_{(A_i, A_j)\in\alpha\times\beta_\rho} f_{ijk}$	$L_1^1(0,\alpha,\beta_\rho)$	$L_2^1(0,\alpha,\beta_\rho)$	$L_3^1(0,\alpha,\beta_\rho)$	μ
$\{A_0,A_1,A_2,A_3,A_4,A_9\}$	(4,2)	15	7	13	$\{A_1\}$
$\{A_0,A_2,A_3,A_4,A_9\}$	(3,2)	13	7	11	$\{A_9\}$
$\{A_0,A_2,A_3,A_4\}$	(1,0)	-	-	-	-

Table 11.8. $\rho = 4$, $\alpha_4 = \{0, 1, 2, 3, 4, 9\}$ *and* $\beta_4 = \{8, 11\}$

α	$\sum_{(A_i, A_j)\in\alpha\times\beta_\rho} f_{ijk}$	$L_1^1(0,\alpha,\beta_\rho)$	$L_2^1(0,\alpha,\beta_\rho)$	$L_3^1(0,\alpha,\beta_\rho)$	μ
$\{A_0,A_1,A_2,A_3,A_4,A_9\}$	(3,2)	15	4	10	$\{A_1\}$
$\{A_0,A_2,A_3,A_4,A_9\}$	(2,2)	13	4	8	$\{A_9\}$
$\{A_0,A_2,A_3,A_4\}$	(1,0)	-	-	-	-

Table 11.9. $\rho = 5$, $\alpha_5 = \{A_0, A_1, A_2, A_3, A_4, A_9\}$ *and* $\beta_5 = \{A_{11}\}$

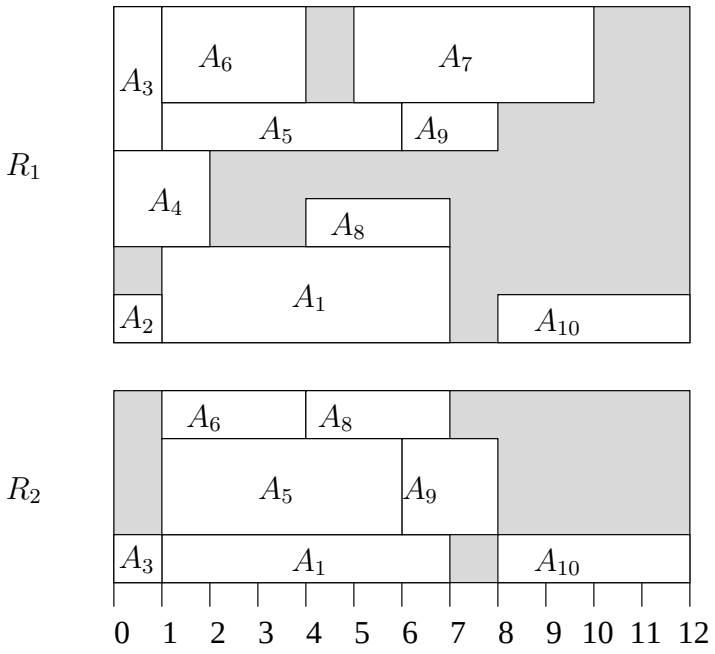


Figure 11.9. *Complete solution after* A_5 *insertion*

11.7. Conclusion

In this chapter, we considered the problem of inserting a single activity A_x inside an existing partial schedule while preserving its structure. The existence of minimum and maximum time lags between activity pair is tackled. The objective is to minimize the project duration increase. This problem consists of selecting the best pair of ordered set (α^*, β^*) inside the resource flow network associated with the existing partial schedule, so that both the flow coming from α satisfies the resource demand of A_x and the insertion does not cause a temporal inconsistency. We have shown how, given a pair (α, β), that the latter constraint can easily be checked by considering three categories of elementary cycles. Moreover, we also show that the project duration increase can also be computed by considering the length of these same elementary cycles.

Whereas the insertion problem is polynomially solvable for the standard RCPSP (where only precedence constraints are taken into account), we showed that the introduction of minimum and maximum time lags makes the problem NP-hard. Nevertheless, when only minimum time lags are considered, the problem turns back polynomially solvable and we exhibit an algorithm to solve it.

Once again, we observe that dealing with maximum time lag constraints is difficult. It was already known that finding a resource-feasible schedule that respects both the minimum and maximum time lags is NP-hard. We saw here that, though apparently easier, even the problem of inserting a single activity in an already existing resource-feasible and time-consistent schedule is NP-hard. Therefore, it might appear relevant in some situations to relax maximum time lags, intending to find a schedule which minimizes the maximum time lag violation (a maximum makespan value being preliminary set). Indeed, in many schedule environments, maximum time lags correspond to user preferences and hence it makes sense to produce a schedule that does not respect them. In this case, one can also define priorities in order to define a hierarchy among maximum time lags. In this case, a possible objective function would be the minimization of the weighted sum of the time lag violations.

Chapter 12

Reactive Approaches

Project scheduling research often focus its attention to providing efficient solutions assuming a static and deterministic environment. However, many disruptions may occur during the execution of a schedule. Several approaches exist for handling those disruptions: we will focus here on what is usually called *predictive-reactive* approaches.

12.1. Dynamic project scheduling

This chapter deals with a project scheduling problem in which disruptions may occur during execution: new incoming orders, order cancellations, missing raw materials, power shortages, machine break-downs, etc. This problem is very general as it includes many scheduling problems, and many types of unexpected events are considered (addition, removal or modification of temporal constraints, resources and activities). We propose a predictive-reactive approach for a situation where no stochastic model of the possible disruptions can be provided. As far as we know, the only work handling this case is a flow-based heuristic proposed by Artigues *et al.* [ART 03] for the addition of unexpected activities in a RCPSP. Chapter 13 addresses an alternative and complementary proactive-reactive approach, aiming at anticipating disruptions through the generation of a robust baseline schedule.

The idea of predictive-reactive approaches is to build a first operational solution that will be re-optimized when an unexpected event is to be handled. Two questions must be addressed:

Chapter written by Christelle GUÉRET, Narendra JUSSIEN.

– **When is a new schedule needed?** A new schedule can be computed each time an unexpected event occurs (e.g. [WU 93] and [BIE 99]); or, at predetermined time intervals [CHU 92]; or, finally, when a given threshold in perturbation is reached [VIE 00]. Generally, rescheduling provides better performance because the overall system reacts quickly to unexpected events. However, a significant amount of computing time can be required, and, moreover, highly varying consecutive solutions may be computed.

– **How is a new schedule to be computed?** Computing a new schedule consists of either computing a new optimal solution or repairing the current unfeasible schedule. Generally, operational techniques repair the current solution in order to maximize the stability of successive solutions [AKT 99, BEA 91, BIE 99, WU 93]. Computing a new optimal solution is seldom applied. Indeed, the re-optimization problem is an NP-complete problem [PIC 95]. The worst-case consists of computing from scratch a new schedule each time an unexpected event occurs. Moreover, generally the successive obtained schedules are quite different one from another and computing time remains quite prohibitive.

In this chapter, a new technique[1] for computing successive optimal stable solutions each time an unexpected event occurs is introduced. This technique uses a recent extension of constraint programming: explanation-based constraint programming.

12.2. Explanations and constraint programming for solving dynamic problems

Constraint programming [TSA 93] is the study of computational systems based on constraints (see also Chapter 4). The idea of constraint programming is to solve problems by stating constraints (conditions, properties) which must be satisfied by the solution. Work in this area can be tracked back to research in Artificial Intelligence and Computer Graphics in the 1960s and 1970s. Only in the last decade, however, has emerged a growing realization that these ideas provide the basis for a powerful approach to programming, modeling and problem solving and that different efforts to exploit these ideas can be unified under a common conceptual and practical framework: constraint programming.

12.2.1. *Dynamic problems and constraint programming*

The notion of *dynamic constraint satisfaction* has been first introduced in [DEC 88]. A *dynamic constraint satisfaction problem* (DCSP) is a sequence of constraint satisfaction problems, each one resulting from some change in the definition of the previous one. These changes may affect any component in the problem definition:

1. This approach has been developed in the PhD dissertation of Abdallah Elkhyari.

variables (addition or removal), domains (value addition or removal), constraints (addition or removal), constraint scopes (variable addition or removal) or constraint definitions (changes in the intensional definition, tuple addition or removal in case of extensional definition), Because domains can be seen as unary constraints, because variables are implicitly added or removed with all the constraints that apply to them, and because any change in a component can be seen as a removal followed by an addition.

Constraint-based techniques for solving dynamic problems can be classified into two categories [VER 05]: proactive methods (which suppose an existing model of the possible modifications) and reactive methods. The latter were addressed with two main approaches:

– solution (complete solutions, partial instantiations, etc.) reuse techniques as in [VER 94] which are based on the assumption that the previous solution is a good basis for finding a solution to the next problem;

– reasoning reuse techniques (as in [SCH 94], or in explanation-based constraint programming [JUS 03]) which prefer to keep track of the portions of the search space that have been set aside in previous resolutions and that can be still set aside in the new problem. The idea behind being to learn from the past in order to avoid future failures.

Several constraint-based reactive scheduling systems arose in the end of the 1980s and in the 1990s [Le 94a]. Collinot and Le Pape [COL 91] introduced the SONIA system in order to compute acceptable solutions to 'evolutive' scheduling problems, i.e., problem whose data may be subject to variation. Their approach relies on an *intelligent backtracking* technique. Burke and Prosser also proposed a Truth Maintenance System (TMS) based approach in the DAS system [BUR 91]. Finally, Smith [SMI 94] introduced the OPIS system based on two notions: conflicts and opportunity for providing repairs and potential improvements of schedules upon handling an unexpected event. Those three systems build a compact description of the contradictions encountered during search and use this information to perform a sophisticated (and therefore not necessarily chronological) backtrack. Unfortunately these approaches are prone to the hazards of search compared to a from-scratch rescheduling as shown by Baker [BAK 94].

12.2.2. *Explanation-based constraint programming*

In constraint programming , the constraint solver's main activity resides in computing redundant information for the constraint network characterizing the solved problem. This redundant information is, most of the time, far from direct and will allow to quickly prune the different possibilities for the variables of the problem. Such an information can be of different kinds: domain reduction (value removal, bound adjustments, etc.), failure identification and new implicit constraint addition. Providing

an *explanation* in this context, consists of *justifying* each new information computed by the solver in order to be able to quickly evaluate the validity of this information in another context. Such a justification is provided as a (precise) set of constraints whose conjunction logically leads to the information being justified. *Explanations* can be seen as an explicit trace of the solver's behavior as the whole set of interactions between constraints that led the solver to provide the new information do appear in the explanation.

Explanations have now clearly shown their interest for constraint programming [JUS 03]: constraint solvers are no longer black boxes only able to answer *yes* or *no* when solving a problem, they now initiate a real interaction with the final user, being able, for example, to explain why a constraint network is unfeasible. Moreover, explanations are quite useful in a dynamic context as they can efficiently and incrementally remove a constraint because they provide a direct access to its past effects [DEB 03]. Finally, and this particularity will also be used for project scheduling, explanations can be used to design new powerful search techniques that will replace classical backtracking with a true repair of the decision path without an excessive loss of past information [GUÉ 00, JUS 02]. This substantial modification in classical search schema allows a fruitful consideration of the information gathered through search: re-exploration of significative portions of the search space is avoided and search, while remaining complete, is failure-driven and can overcome the somewhat arbitrary exploration induced by backtracking techniques.

Using an explanation-based constraint solver induces two prerequisites:

– any used constraint needs to be able to explain any of the information it provides;

– the solving technique needs a branching scheme that involves basic (i.e. available *as is* in the solver's library of constraints) constraints whose negation is also a basic constraint. Indeed, in explanation-based constraint programming, decisions (traditionally a variable-value assignment) and their negations (traditionally, a value removal) are explicitly handled as opposed to traditional backtracking-based approach where all negations are implicit (thanks to backtracking).

In the following, we will actively use the capability of such a system to both take into account a modification of the problem definition and to explain past decisions in order to compute a new optimal solution when an unexpected event occurs.

12.3. An explanation-based approach for solving dynamic project scheduling

Our system is based upon:

– a tree-based search embedded within a constraint solver: in each node, deduction rules are applied in order to propagate the constraints of the problem[2];

– the active use of *explanations*. They are provided by the constraints used for solving the RCPSP and will be used to dynamically handle unexpected events. Indeed, as already mentioned, handling an unexpected event leads to remove, add, or modify a constraint. Upon removing a constraint, a re-optimization is launched from the previous feasible solution of the problem. For the two other cases, if the current solution is not anymore valid, explanations can be of some help to determine constraints responsible for the contradiction. The re-optimization is performed taking into account the fact that at least one of these constraints needs to be removed (adding, for that, its negation).

We first present the explanation-based tree-search algorithm for solving static RCPSP and then explain how to extend it to solve dynamic instances.

12.3.1. *The tree search to solve static instances*

The tree search we use is largely inspired from the one from Brucker *et al.* [BRU 98] and previously described in section 5.2.1 (see also section 6.1.3).

Each node in the search tree is defined by three disjoint sets: a set C of pairs of activities that are in conjunction, a set P of pairs of activities that are overlapping, a set F of pairs of activities that are said to be flexible:

– $C = \{(i,j)|(i \rightarrow j) \vee (j \rightarrow i)\}$;

– $\phi = \{(i,j)|(i \parallel j)\}$;

– $U = \{(i,j)|(i,j) \notin C \wedge (i,j) \notin P\}$.

Branching consists in transferring a pair of activities from F to C (thus imposing $i \rightarrow j$ or $j \rightarrow i$) or to ϕ (imposing $i \parallel j$). This branching is performed until U becomes empty. In each node, propagation rules add new implicit constraints and therefore complete sets C and ϕ while removing pairs from U. Backtracks and repairs made on the set of decisions taken so far may transfer back some pairs of activities from C or ϕ to U.

As already mentioned, our technique relies on the fact that decisions taken during search do posses an easily stated negation. However, for example, negating $i \rightarrow j$ leads to the disjunction $j \rightarrow i \vee i \parallel j$, and disjunctions adding non-determinism in

2. In the following, the word *constraint* will refer not only to initial constraints of the problems (precedence and resource constraints), but also to each decision made in the branching scheme and to each deduction made by propagation rules during search.

a problem are generally poorly handled in constraint programming. This is why we introduced the notion of *distance*.

DEFINITION 12.1.– *Let A_i and A_j be two distinct activities. The distance d_{ij} between A_i and A_j is the time lag between the end of activity A_i and the beginning of A_j. Therefore, we have: $d_{ij} = S_j - S_i - p_i$.*

Stating sequencing constraints using this notion of distance is easy and leads to the relations reported in Table 12.1.

Decision	Distance constraint	Negation	Distance constraint
$i \rightarrow j$	$d_{ij} \geq 0$	$i \not\rightarrow j$	$d_{ij} < 0$
$j \rightarrow i$	$d_{ij} \leq -p_i - p_j$	$j \not\rightarrow i$	$d_{ij} > -p_i - p_j$
$i \parallel j$	$d_{ij} \cdot d_{ji} > 0$	$i \leftrightarrow j$	$d_{ij} \cdot d_{ji} \leq 0$

Table 12.1. *Decisions and their negation expressed using the distance*

Our approach based on the general MAC-DBT algorithm [JUS 00] can be seen as a generalization of the intelligent backtracking technique introduced in [COL 91] because the information gathered through search is used as much as possible by replacing a traditional backtrack with a real repair in order to save as much reasoning as possible that is reusable in the new context.

12.3.2. *Incrementally handling unexpected events*

The key point of our approach is that search is considered as a dynamic problem (new decisions are considered as constraint additions and backtracks or repairs as constraint removals) leading to a solving technique able to handle any new event considered as a constraint addition or removal.

Our system is able to incrementally handle various events:

– **Temporal events**: adding, removing or modifying sequencing constraints (precedence, disjunction, overlapping between two activities including their generalized cases – taking into account a minimal distance between activities). The events may be related to the incoming of new orders, or new technical constraints.

– **Activity-related events**: adding, removing or modifying an activity. Those events may be related to the incoming of new orders, removal of others, modification of their duration or modification of their resource needs, etc.

– **Resource-related events**: adding, removing or modifying a resource. The events may be related to machine break-down, machines coming back after maintenance, etc.

Once a first solution is computed with the tree search for the initial problem, modifications are incrementally taken into account in the following way:

– upon *adding* a new information: if the current solution is still feasible, nothing is needed to be done; however, if a contradiction is introduced by the new information a re-optimization step is performed. This re-optimization starts from the current solution and uses the explanations provided by the solver to, first, determine the set of decisions responsible for this contradiction, and, second, to dynamically remove from this set one or more decisions in order to compute a new solution;

– upon *removing* an information: re-optimization is asked after having performed the incremental removal [DEB 03] of all the constraints related to this information;

– upon *modifying* an information: a removal (of the old information) followed by an addition (of the new information) is performed.

Notice that adding or modifying constraint may lead to an over-constrained instance (for which no solution exists). Thanks to explanations, our system is able to provide a contradiction explanation that can be used by the user to select a constraint to be removed or modified, or to allow a degradation of the makespan in order to compute a new solution. Several tools can be designed to help the user to make his choice in this context [JUS 01].

12.4. Experimental results

As no other technique exists for optimally solving RCPSP instances in a dynamic context, we compared our technique to a rescheduling from scratch after each event. As no real benchmark has been defined for this problem, we created benchmarks based upon existing instances for the static case. We present here[3] results both in terms of execution time and of stability of successive solutions for adding and removing a set of activities.

12.4.1. *Stability measures*

Performance of dynamic scheduling systems can be measured both using computing time and quality of the obtained solutions. The quality of successive solutions can be measured comparing the new solution to the previous one or to the initial one. We evaluated the performance of our approach by studying the evolution of this quality over time.

3. Those results mainly come from the PhD dissertation of Abdallah Elkhyari [ELK 03a].

Several stability measures were used: number of modified starting times, sum of the modifications, maximal modification, etc. and finally, the number of relative positions between activities that have been modified. This is the measure reported in the following as it seems the most relevant measure for scheduling problems. **POSI(**S_p**,** S_q**)** represents the number of relative positions that have changed when switching from schedule S_p to schedule S_q. POSI$(S_p, S_q) = \sum_{(i,j)\in A\times A,\ i<j} \gamma_j^i(S_p, S_q)$ where $\gamma_j^i(S_p, S_q) = 1$ if the relative position of A_i and A_j has changed and 0 otherwise.

12.4.2. *Test set*

We use a data set generated thanks to *ProGen* from Kolisch *et al.* [KOL 95b]. We use the SMCP (*Single-Mode Project Scheduling*) series. In the following, P_0 denotes the original problem of the series and P_i the problem obtained after the i^{th} perturbation.

Problems used for studying the consequences of the addition of a series of activities are built in the following way:

– the initial problem P_0 is a SMCP problem from which activities are removed, as well as the associated precedence constraints. In order to be sure that P_0 is still a RCPSP, precedence constraints between the predecessors of the activity removed and dummy activity A_0, and between its successors and dummy activity A_{n+1} are added;

– problem P_i is obtained from problem P_{i-1} by randomly adding one activity among the ones that have been initially removed. Precedence constraints that had been removed with this activity and that link it with other activities of problem P_{i-1} are also added.

In order to study the consequences of the removal of a series of activities, each series of problems are constructed as follows:

– the initial problem P_0 is a SMCP problem;

– problem P_i is obtained from problem P_{i-1} by randomly removing one activity and its precedence constraints, and by adding the precedence constraints that connect its successors and predecessors to the dummy activities.

A large number of problems have been generated for each type of event. Thus, for the addition and the removal of a series of activities, 10 series of 10 problems each have been constructed for each following size: 12 activities and 4 resources, 22 activities and 4 resources, 32 activities and 1 resource, 32 activities and 2 resources, 32 activities and 3 resources, and 32 activities and 4 resources. For coherence matters, the number of additions/removals of activities depends on the size of the problem: 2 for problems with 12 activities, 3 for problems with 22 activities and 4 for the others.

12.4.3. *Computational results*

We only present here the results obtained for the addition and removal of activities. The results are quite similar for the addition and the removal of precedence or generalized temporal constraints.

In the following, `ppce` represents our approach, and `ropt` corresponds to a rescheduling from scratch.

Let $T_{\texttt{ppce}}(P_k)$ (respectively $T_{\texttt{ropt}}(P_k)$) be the time needed (in seconds) to compute a solution for problem P_k with our system (respectively by a rescheduling from scratch). The gain in time is defined by:

$$G_{time}(P_i) = 1 - \frac{\sum_{0 \leq k \leq i} T_{\texttt{ppce}}(P_k)}{\sum_{0 \leq k \leq i} T_{\texttt{ropt}}(P_k)}$$

The average gain in stability of our approach compared with a rescheduling form scratch is:

$$G_{\text{POSI}}(P_i, P_j) = 1 - \frac{\sum_{1 \leq k \leq s} \text{POSI}(S_i^k(\texttt{ppce}), S_j^k(\texttt{ppce}))}{\sum_{1 \leq k \leq s} \text{POSI}(S_i^k(\texttt{ropt}), S_j^k(\texttt{ropt}))}$$

where $S_i^k(\texttt{ropt})$ is the solution of the i^{th} problem of series k obtained by a rescheduling from scratch, and $S_i^k(\texttt{ppce})$ is the solution obtained by our approach.

12.4.3.1. *Analysis of the results in terms of `cpu` time*

The average execution time obtained for these problems varies from 2.5 s to 61 s for the addition of activities, and from 5 s to 66 s for the removal of activities. Tables 12.2 and 12.3 present the gain in time for these two types of unexpected event. These tables contain : the minimal (**Min**), maximal (**Max**) and mean (**Mean**) gain for each step of the series, and for all problems of the series. The value (**AvG**) represents the average gain between the mean execution time of our approach to compute all the successive solutions of the series, and the mean execution time of the traditional approach.

Table 12.2 summarizes the results obtained for the resolution of problems in which a new activity is added at each step. The mean gain (**Mean**) is between 10.7% and 78.2%, and the average gain **AvG** is very important for all problems: between 43.7% and 90.5%, the average gain seems to increase with the size of the problems.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		
	Min/Max	Mean	Min/Max	Mean	Min/Max	Mean	Min/Max	Mean	AvG
12×4	-0.9%	37.7%	-63.8%	32.2%	–	–	–	–	43.7%
	64.4%		69.6%		–		–		
22×4	-34.6%	29.7%	-208%	5.8%	-155.2%	34.4%	–	–	78.9%
	94.4%		94.4%		96%		–		
32×1	30%	58.3%	18.2%	68.5%	40.7%	77%	-66.1%	46.7%	83.8%
	98.7%		99.3%		98.7%		99.6%		
32×2	-280.9%	10.7%	-140.1%	57.8%	-64.2%	68.7%	-78.9%	62.8%	84.9%
	91.2%		97.1%		96.9%		97.6%		
32×3	12.7%	61.4%	15.9%	66%	16.8%	74.6%	30.6%	78.2%	90.5%
	95.2%		98.2%		98.8%		96.3%		
32×4	33.3%	57%	-18.1%	63.5%	-19.6%	60.7%	26.7%	78.2%	90%
	75.9%		97.4%		98.7%		99.2%		

Table 12.2. *Gain in time for the addition of a series of activities.*

Tables 12.3 contains the results obtained for the resolution of series of problems from which one activity is removed at each step. The mean gain is between -51% and 65.5%, and the average gain *AvG* varies from -19.1% and 59.4%. Again, the gains are important for series of large problems, less important for series of average size, and we can note some losses for some smaller problems.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		
	Min/Max	Mean	Min/Max	Mean	Min/Max	Mean	Min/Max	Mean	AvG
12×4	-212.2%	-51%	-195.5%	-65.3%	–	–	–	–	-19.1%
	21.7%		19.5%		–		–		
22×4	-637.8%	-22.9%	-242.8%	20.5%	-232.1%	21.5%	–	–	44.7%
	91.5%		94.3%		88.8%		–		
32×1	-185%	13.4%	-119%	20.6%	-102.6%	0.9%	-151.5%	-0.5%	47.7%
	97.1%		92.7%		91.1%		91.5%		
32×2	-148.4%	24.2%	-219.4%	14.4%	-161.2%	38.9%	-164.8%	34%	59.4%
	98%		98.7%		97.4%		97.5%		
32×3	0%	65.5%	-34.8%	52.3%	-38.3%	48.1%	-43.1%	49.2%	51.8%
	94.1%		94%		91.7%		90.3%		
32×4	-280.4%	31%	-259.5%	32.5%	-180.2%	28.3%	-148.1%	26.3%	59.4%
	92.4%		95.7%		92.3%		91.3%		

Table 12.3. *Gain in time for the removal of a series of activities.*

12.4.3.2. *Analysis of the stability*

The following tables contain, for each type of unexpected event, the mean gain in stability G_{POSI} presented earlier.

Table 12.4 summarizes the results obtained for the series of problems in which one activity is added at each step. The mean gain is between -51.8% and 81%. The results show an important gain for problems with 32 activities. One more time, the technique proposed is more efficient for larger problems than smaller ones.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-24%	-51.8%	–	–	-58.3%
22×4	18.2%	12.5%	-14.6%	–	-19%
32×1	73.1%	79%	81%	-27.6%	11.2%
32×2	53.7%	37.4%	38.2%	63.5%	37.5%
32×3	54.3%	45.8%	76.5%	33.3%	21.1%
32×4	36.7%	62.1%	34.1%	49.7%	27.3%

Table 12.4. *Gain in stability of the* POSI *measure for the addition of a series of activities.*

Table 12.5 contains the results obtained for the resolution of series of problems in which one activity is removed at each step. The mean gain varies from -27% to 43.6%. The gains are important except for the last series. In our approach, the dynamic removal of activities seems to be more efficient concerning stability measure, than the dynamic addition of activities. This is certainly due to the fact that as the problems tested are strongly disjunctive, the addition of a new activity often implies a lot of modifications in the solution.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-27%	1.7%	–	–	28.9%
22×4	17%	25.4%	-7.9%	–	25%
32×1	14.3%	7.2%	15.6%	30%	27.5%
32×2	35.7%	17.9%	-0.1%	39.1%	38.5%
32×3	-0.1%	27.7%	14.9%	43.6%	32.1%
32×4	12.9%	36.5%	20.2%	18.9%	0.9%

Table 12.5. *Gain in stability for the* POSI *measure for the removal of a series of activities.*

12.5. Conclusion

In this chapter, a system dedicated to the resolution of dynamic RCPSP has been presented. This system combines techniques from operations research as well as constraint programming with explanations. The experimentation done on a series of

problems constructed from benchmarks of previous works show that this approach is very efficient in terms of computation time and stability compared with the traditional approach which consists of recomputing a solution from scratch despite the NP-complete nature of this problem.

These first experiments show the interest of an approach based on explanations to take into account unexpected events while preserving solutions close to each others. For very large size problems for which it is not always possible to compute optimal solution, this approach can also be applied to an initial heuristic solution and be used to find a new feasible solution when a random event occurs, instead of searching for the optimal solution.

Our objective now is to improve our system, for example by integrating more efficient deduction rules to handle resource constraints, by proposing more interaction between the system and the user, and by adapting it to more specific scheduling problems (like we have already done for timetabling problems [ELK 03b]).

Chapter 13

Proactive-Reactive Project Scheduling

13.1. Introduction

The vast majority of the project scheduling research efforts over the past several years has concentrated on the development of workable baseline schedules, assuming complete information and a static and deterministic environment. Such a *baseline schedule* (*pre-schedule*, *predictive schedule*) is traditionally constructed by solving the RCPSP. This problem (problem $m, 1|cpm|C_{\max}$ in the notation of [HER 00]) involves the determination of activity starting times that satisfy both the zero-lag finish-start precedence constraints and the renewable|(resource constraints under the objective of minimizing the project duration (for reviews, see the first part of the present book along with [BRU 99, DEM 02b, HER 98a, KOL 99a, KOL 99b]).

During execution, however, a project may be subject to considerable uncertainty, which may lead to numerous schedule disruptions. Many types of disruptions have been identified in the literature (see [YU 04, WAN 05, ZHU 05]). Activities can take longer than primarily expected, resource requirements or availability may vary, ready times and due dates may change, new activities may have to be inserted ([ART 00]), etc. In this chapter we focus on schedule disruptions that may be caused by uncertainty in the duration of the project activities and/or in the availability of the renewable resources.

Proactive/reactive project scheduling procedures try to cope with schedule disruptions that may occur during project execution through the combination of a *proactive*

Chapter written by Erik DEMEULEMEESTER, Willy HERROELEN, Roel LEUS.

scheduling procedure for generating a robust baseline schedule with a *reactive* procedure that is invoked when a schedule breakage occurs during project execution and the schedule needs to be repaired. By contrast, Chapter 12 addresses a predictive-reactive approach where all the effort is concentrated on the repair procedure, while the baseline schedule has not been constructed with a robustness objective.

Research in proactive/reactive project scheduling is growing steadily (for surveys see [AYT 05, HER 04, HER 05, VIE 03]). The objective of this chapter is to discuss promising proactive/reactive scheduling procedures that may be deployed at the occurrence of disruptions that are caused by uncertain activity durations or uncertain resource availabilities.

The remainder of this chapter is organized as follows. In the next section we introduce the solution robust project scheduling problem under activity duration uncertainty and discuss proactive and reactive scheduling policies. Section 13.3 then presents proactive and reactive procedures for dealing with disruptions that may be caused by resource breakdowns. Section 13.4 offers conclusions and suggestions for further research.

13.2. Solution robust scheduling under activity duration uncertainty

13.2.1. *The proactive/reactive scheduling problem*

The problem used as our vehicle of analysis can be described as follows. A project network $G(V, E)$ is represented in activity-on-the-node representation with dummy start node A_0 and dummy end node A_{n+1}. All non-dummy project activities have stochastic activity durations P_i and are subject to zero-lag finish-start precedence constraints on the elements of E: we require $S_j \geq S_i + P_i$ if $(A_i, A_j) \in E$, with S_i the activity start times realized during project execution. Non-dummy activities require an integer per period amount b_{ik} of one or more renewable resource types R_k, $k = 1, \ldots, q$, during their execution. All resource types R_k have a per-period capacity B_k. The activity weight w_i denotes the marginal cost of a deviation between the realized start time S_i of activity A_i and its planned start time s_i in the baseline schedule. These weights may include unforeseen storage costs, extra organizational costs, costs related to agreements with subcontractors or also just a cost that expresses the dissatisfaction of employees with schedule changes. We assume that $w_0 = 0$, while w_{n+1} denotes the cost of delaying the project completion beyond a predetermined deterministic due date d_{n+1}.

The proactive scheduling objective is to build a *solution robust* or *stable* precedence and resource feasible baseline schedule, the activity starting times of which are denoted by s_i. We assume in this chapter that stability is strived for by minimizing the stability function $\sum_{A_i \in V} w_i E\left(|S_i - s_i|\right)$, the weighted sum of the expected absolute

deviation between the realized activity start times S_i and the planned activity start times s_i (E is the expectation operator). Unfortunately, the evaluation of the objective function $\sum_{A_i \in V} w_i E\left(|S_i - s_i|\right)$ is a cumbersome task: [HAG 88] has shown for projects without resource constraints that even when every activity duration P_i can take only two discrete values, then computing the expected project duration and computing the probability that the project is finished by a given time instant, assuming an early-start schedule, is $\#\mathcal{P}$-complete. In our applications, we compute the stability cost through simulation. Finally, it should be mentioned that other stability objective functions and surrogate stability measures have been proposed in the literature ([Van 06a]).

13.2.2. *Proactive scheduling*

Several strategies can be followed for generating stable baseline schedules; in this chapter we assume that a *two-phase procedure* is used. In the first phase, an input schedule is generated that is both precedence and resource feasible but is not intentionally protected against anticipated disruptions. Such a schedule can be generated by solving the underlying RCPSP (problem $m, 1|cpm|C_{\max}$), using (deterministic) single-point estimates p_i for each P_i.

Two strategies may then be used in the second phase to increase the stability of the input schedule. One way is to aim at a robust resource allocation, i.e. to decide on a clever way in which the various resource units are transferred between the activities of the schedule. Another is to insert time buffers that should prevent as much as possible the propagation of distortions throughout the schedule.

Under the assumption that a single activity may deviate from its pre-schedule duration (without knowing which) and a single disruption scenario per activity, [LEU 05] showed that the machine scheduling problem for stability is strongly $\mathcal{NP}$-hard for a single machine with unequal ready times or precedence constraints and for the case of a free number of parallel machines; the single-machine problem without ready times and precedence constraints is still ordinarily $\mathcal{NP}$-hard.

13.2.2.1. *Robust resource allocation*

[LEU 03] and [LEU 04] study the problem of generating a *robust resource allocation* for a *given* feasible baseline schedule. They explore the fact that the search for an optimal resource allocation reduces to the search for a resource flow network (which describes the routing of resources across the activities in the schedule, see section 1.6) that exhibits desirable robustness characteristics. A branch-and-bound algorithm is developed that solves the robust resource allocation problem in exact and approximate formulations for the case of a single resource type and exponential activity duration disruption lengths. For the general multi-resource type case, Deblaere *et al.* [DEB 06b] derive lower bounds on scheduling stability and propose three integer

programming based resource allocation heuristics and one constructive procedure that perform well on the J30, J60 and J120 network instances of the PSPLIB problem set (see [KOL 97] and Chapter 7).

13.2.2.2. *Buffer insertion*

[Van 05, Van 06a, Van 08a, Van 06b, Van 06c] have developed several exact and suboptimal procedures for inserting time buffers in an input schedule under the objective of minimizing the stability cost function $\sum_{A_i \in V} w_i E\left(|S_i - s_i|\right)$. Among them, the *starting time criticality* (STC) *heuristic*, despite the simplicity of its underlying assumptions and structure, obtained excellent results.

The STC-heuristic exploits information about both the weights of the activities and the variance of the activity durations. The basic idea is to start from a precedence and resource feasible but unbuffered input schedule and iteratively create intermediate schedules by inserting a unit-time buffer in front of the activity that is the most starting time critical in the current intermediate schedule, until additional safety time no longer improves stability. The starting time criticality of an activity A_j is defined as $stc(A_j) = Pr[S_j > s_j] \times w_j = \gamma_j \times w_j$, where γ_j denotes the probability that activity A_j cannot be started at its scheduled starting time s_j.

At each iteration step (see pseudo-code in Figure 13.1) the buffer sizes of the current intermediate schedule are updated as follows. The list is scanned in non-increasing order of $stc(A_j)$ and the size of the buffer to be placed in front of the currently selected activity from the list is augmented by one time period (with appropriate updates of the starting times of the direct and transitive successors of the activity). If this new schedule has a feasible project completion ($s_{n+1} \leq d_{n+1}$) and results in a lower estimated stability cost ($\sum_{A_j \in V} stc(A_j)$), the schedule serves as the input schedule for the next iteration step. If not, the next activity in the list is considered. When an activity A_j is reached for which $stc(A_j) = 0$ (all activities A_j with $s_j = 0$ are by definition in this case), the procedure terminates.

Regrettably, the probabilities γ_j are not easy to compute. STC makes two assumptions in approximating γ_j: (a) predecessor A_i of activity A_j starts at its originally planned starting time when calculating the event that predecessor A_i disturbs the planned starting time of activity A_j, and (b) only one activity at a time disturbs the starting time of activity A_j.

13.2.3. *Reactive scheduling*

Proactive-reactive scheduling implies that the buffered baseline schedules generated by the proactive procedures should be combined with reactive scheduling procedures that are deployed during project execution when disruptions occur that cannot be absorbed by the baseline schedule.

```
calculate all stc(A_i)
sort activities by non-increasing stc(A_i)
while no improvement found do
    take next activity A_j from list
    if stc(A_j) = 0 then procedure terminates
    else add buffer in front of A_j
        update schedule
        if improvement & feasible then
            store schedule
            goto next iteration step
        else
            remove buffer in front of A_j
            restore schedule
```

Figure 13.1. *The iteration step of the STC heuristic.*

The literature concerning reactive project scheduling is virtually void. [YU 04] describe an ILP model for the Multi-Mode RCPSP (problem $m, 1T\,|cpm, disc, mu|\,C_{\max}$ in the notation of [HER 00]) and report on computational results obtained by a hybrid mixed integer programming/constraint propagation approach for minimizing the schedule deviation caused by a single disruption induced by a known increase in the duration of a single activity. [Van 06c] developed and extensively evaluated a number of exact and heuristic reactive procedures.

The reactive scheduling problem at decision point t when the baseline schedule breaks can be viewed as a RCPSP with weighted earliness tardiness costs (problem $m, 1\,|cpm|\,early/tardy$ in the notation of [HER 00]). Due dates are set equal to the activity completion times s_i+p_i in the predictive schedule. The earliness and tardiness costs may be assumed to be symmetrical and chosen as the weights w_i in the stability objective function, with a possible exception for the earliness cost of the dummy end activity, which can be set equal to zero.

Efficient exact procedures for solving problem $m, 1\,|cpm|\,early/tardy$ have been proposed in the scheduling literature ([SCH 00b, VAN 01a, KÉR 05]). However, [Van 08a] found that calling an exact weighted earliness-tardiness procedure at each schedule breakage point becomes computationally infeasible already for small networks. [Van 06a] obtained excellent computational results with a sampling approach and an adapted version of the local search algorithm of [BAL 07]. The sampling approach requires less computational time for similar stability cost performance.

The basic *sampling approach* ([Van 06a]) relies on different priority lists in combination with different schedule generation schemes. It tries to make a suitable decision at each decision time t as illustrated in Figure 13.2. *Step* 1 checks for new information

for $t = 0, \ldots, H$ **do**
Step 1: check for new scheduling information
Step 2: **if** no new information **then** $S^t = S^{t-1}$ and goto period $(t+1)$
else goto *Step* 3
Step 3: **for** list $l = 1, \ldots, L$ **do**
construct $S^t_{\lambda_l,RP}$ and calculate $\Delta(S^0, S^t_{\lambda_l,RP})$
construct $S^t_{\lambda_l,RS}$ and calculate $\Delta(S^0, S^t_{\lambda_l,RS})$
construct $S^t_{\lambda_l,P}$ and calculate $\Delta(S^0, S^t_{\lambda_l,P})$
construct $S^t_{\lambda_l,S}$ and calculate $\Delta(S^0, S^t_{\lambda_l,P})$
store the schedule S^t that minimizes $\Delta(S^0, S^t)$
Step 4: start all activities A_i with $s^t_i = t$.

Figure 13.2. *Description of the basic sampling approach.*

becoming available at time t. If at time t no activity finishes and no activity was projected to finish, then no new information became available since the previous decision point $(t-1)$. The previous schedule S^{t-1}, generated at time $(t-1)$, remains valid (*Step* 2). Instead of using one priority list in combination with one schedule generation scheme, *Step* 3 uses multiple lists $\lambda_l \in \{\lambda_1, \ldots, \lambda_L\}$ at time t in combination with various schedule generation schemes.

[Van 06a] evaluate different priority lists (EBST (earliest start time in the baseline schedule), LST (latest starting time), LW (largest activity weight), LAN (lowest activity number), RND (random), EPST (earliest starting time in the schedule generated at the last decision point), MC (lowest current stability cost)). For each of these priority lists λ_l, a complete schedule is generated using four schedule generation schemes.

The parallel schedule generation scheme ($S^t_{\lambda_l,P}$) iterates over time and starts at each decision point, in the order dictated by the priority list, as many unscheduled activities as possible in accordance with the precedence and resource constraints (see Chapter 6). The robust parallel schedule generation scheme ($S^t_{\lambda_l,RP}$) is similar to the parallel scheme but considers at each decision time t only the activities for which the current decision time t is greater than or equal to their planned starting time in the baseline schedule. The serial schedule generation scheme ($S^t_{\lambda_l,S}$) schedules at each decision point t the next activity from the priority list (see Chapter 6). The robust serial schedule generation scheme ($S^t_{\lambda_l,RS}$) considers the activities in the order dictated by the priority list and starts them at a feasible time as close as possible to their planned starting time in the baseline schedule.

In this way, a total of $4 \times L$ candidate schedules are generated and the schedule $S^t_{\lambda_l,.}$ yielding the smallest stability cost deviation $\Delta(S^0, S^t_{\lambda_l,.})$ from the baseline schedule S^0 is stored. The procedure then continues in *Step* 4 by starting the activities for which the planned starting time in the schedule equals t.

13.3. Solution robust scheduling under resource availability uncertainty

The literature on proactive/reactive project scheduling under resource availability uncertainties is virtually non-existant. [DRE 05] considers the problem of project planning subject to human resource constraints which have to do with job competences, working hour limits, vacation periods and unavailability of employees. She presents a mathematical model as well as dedicated algorithms for robust schedule generation and schedule repair. [YU 04], in their above-mentioned ILP model for the multi-mode problem, allow for (known) decreases in the resource availabilities in certain planning periods.

13.3.1. *The problem*

Contrary to section 13.2, activity durations are now assumed to be deterministic; uncertainty originates from the stochastic nature of renewable resource availability $\mathcal{B}_k$ ($k = 1, \ldots, q$). This means that during schedule execution infeasibilities may occur due to renewable resource breakdowns, causing the schedule to need repair. The proactive project scheduling problem then consists of generating a proactive schedule that is protected as well as possible from such disruptions, subject to a project deadline, finish-start, zero-lag precedence constraints and renewable resource constraints. A deterministic maximum resource availability B_k is considered for each renewable resource type R_k. Each of these B_k resource units initially allocated to the project is subject to breakdowns; some of the proposed models pre-suppose knowledge of the mean time to failure and the mean time to repair. The objective function to be minimized is again $\sum_{A_i \in V} w_i E\left(|S_i - s_i|\right)$, the weighted expected deviation between the planned and actually realized activity start times.

[LAM 06] develop and evaluate eight proactive and three reactive scheduling procedures; one of the hypotheses that lie at the basis of their computations is that during project execution activities are not initiated before their baseline starting time. The best proactive and reactive strategies can be described as follows.

13.3.1.1. *Proactive strategy*

The best performing proactive strategy consists of the addition of *time buffers* to a schedule which is generated using *resource buffering*, and which places the activities with highest impact on total project stability earliest in time (*largest CIW first*).

Resource buffering boils down to planning the project subject to a deterministic nominal resource availability that is strictly below B_k. More precisely, the nominal availabilities are set equal to $E(\mathcal{B}_k) = \sum_{m=0}^{B_k} (B_k - m)\pi_m$, where π_m denotes the steady state probability that m resource units of resource type R_k are inactive. When necessary, this value is increased to $\max_{A_i \in V} b_{ik}$ to allow for the activity with the highest resource demand for resource type R_k to be scheduled.

The largest CIW first rule schedules the activities A_i in non-increasing order of their cumulative instability weight $CIW_i = w_i + \sum_{A_j \in \Gamma_i^*} w_j$, where Γ_i^* denotes the set of direct and indirect successors of activity A_i. In the first step a precedence feasible priority list is constructed in which precedence-unrelated activity pairs appear in non-increasing order of CIW_i (lowest activity number as tie-breaker). Afterward this priority list is transformed into a precedence and resource feasible schedule using a serial schedule generation scheme .

Time buffering implies that time buffers are inserted in front of activities in order to absorb potential disruptions caused by earlier resource breakdowns and the resulting activity shifts. The procedure approximates Δ_i, the duration increase in activity A_i caused by resource breakdowns, by $\max_{R_k \in \mathcal{R}}\{\frac{p_i}{MTTF_k} MTTR_k r_{ik}\}$, where p_i denotes the expected duration of activity A_i, and $MTTF_k$ and $MTTR_k$, respectively denote the mean time to failure and mean time to repair for resource type R_k. The impact on the objective function that can be attributed to the disruption of activity A_j caused by its predecessors $i \in \Gamma_j^{*-1}$ is then estimated as $I_j = \sum_{A_i \in \Gamma_j^{*-1}} w_j \max\{0, s_i + p_i + \Delta_i - s_j\}$. All non-dummy activities are placed in a list L ordered by non-increasing impact estimates, with the lowest activity number as tie-breaker. The first activity in list L is selected and right-shifted by one time unit. Affected activities are likewise right-shifted by one time unit in order to keep the schedule precedence and resource feasible. In case the resulting schedule also respects the project deadline constraint, the procedure moves to the next iteration by recalculating the expected impacts for each activity in the new schedule and building a new list L'. In case the new schedule violates the deadline, the move is reverted and the procedure examines the next activity in the list. If no such activity can be found, the procedure terminates.

13.3.1.2. *Reactive strategy*

When the baseline schedule breaks, i.e. when activities have to be interrupted because of a resource breakdown, the schedule needs to be repaired using a reactive procedure. Lambrechts *et al.* [LAM 06] investigate a preempt-repeat setting (interrupted activities have to be restarted anew); they generate a list L containing all activities that are not yet completed at the time of interruption, ordered in non-decreasing order of the baseline starting times. This list is then decoded into a feasible schedule using a serial schedule generation scheme that tries to schedule the interrupted activities as early as possible; a tabu search algorithm to improve the generated reactive schedule is also proposed in the same source.

13.4. Conclusions and suggestions for further research

Real-life projects are typically subject to considerable uncertainty. This chapter has addressed proactive/reactive project scheduling procedures that may be deployed

when the uncertainty pertains to the duration of activities or to the availability of renewable resources. *Proactive* procedures have been described to generate a robust baseline schedule that is appropriately protected against distortions that may occur during project execution. The term 'robustness' in this context refers to solution robustness or stability. Our aim has been to generate proactive precedence and resource feasible baseline schedules that minimize one particular stability cost function, namely the weighted sum of the expected deviation between the actually realized activity start times during project execution and the planned activity start times in the baseline. When distortions during project execution cause the baseline schedule to become infeasible, a reactive policy needs to be invoked to repair the schedule; this chapter has (to a lesser extent) also provided insights into this *reactive* issue.

For variable activity durations, [Van 05, Van 06a, Van 08a, Van 06b, Van 06c] have developed exact and heuristic proactive time buffer insertion strategies that can be combined with effective reactive policies for (optimally or heuristically) solving the underlying weighted earliness-tardiness problem. These research efforts allow for drawing interesting and reassuring conclusions. It appears that the combination of proactive and reactive scheduling techniques leads to significant stability improvements decreases in the planning nervousness, with only moderate (hence acceptable) increases in schedule makespan.

The unavailability of resources is a second potential but very realistic cause of substantial deviations from the baseline schedule. Consequently, the development of proactive/reactive scheduling procedures under stochastic resource availability is relevant from a theoretical and a practical point of view. Research in this area is just emerging. [LAM 06] have obtained excellent results with a dedicated procedure that combines a resource and time buffered proactive scheduling procedure with a list based reactive schedule generation scheme .

These promising results justify the engagement in additional research. The development of a single-step (monolithic) proactive scheduling procedure for the generation of stable (solution robust) baseline schedules with acceptable makespan performance, which can be easily combined with effective reactive scheduling policies that are able to operate under various types of schedule distortions, deserves priority. The exploration of robustness measures other than the weighted activity starting time deviations, which was explored in this chapter, constitutes an other interesting area of future research.

13.5. Acknowledgements

This research has been supported by Project OT/03/14 of the Research Fund K.U. Leuven, Project G.0109.04 of the Research Foundation – Flanders (F.W.O.-Vlaanderen) and Project 06163 supported by the National Bank of Belgium.

Chapter 14

RCPSP with Financial Costs

A project is a complex system in which many different persons and several means and actions interact to give an answer to a client's request. As an answer, it has to respond to the client's wishes in terms of time, quantity and costs. These three objectives are often incompatible and traditionally only one of them is taken into account. For the last 10 years, the project scheduling problem with makespan minimization has been the subject of several papers. The studies on project scheduling with financial costs received a little less interest. In this chapter, we focus on the way that financial aspects of project scheduling is treated in the studies. This chapter is inspired by [DRE 02] and [DRE 03].

14.1. Problem presentation and context

As shown in this book, the project scheduling problem is characterized by activities, resources and an objective function. All activities of a project have to be completed to achieve the project. Project activities have one or several execution modes (see Chapter 9) associated with a duration and require the use of resources (this requirement can be constant or not). Activities can be linked with precedence relations representing technological constraints and can also be associated with time windows (release time, due dates or deadlines). Associated with an activity, a cash flow may occur. To process activities, the project needs resources (workforce, capital, machine etc.). Resources used by the project can be of four different types: renewable, non-renewable (as capital in special cases), doubly constrained and partially renewable.

Chapter written by Laure-Emmanuelle DREZET.

Costs can be caused by resources or by activities. When costs are generated by activities, the way activities are processed (execution mode, starting or finishing time) generates costs that have to be minimized. When costs are generated by resources, the scheduling of activities indirectly impacts on the budget of the project via the resource. When costs are incurred by activity execution modes, we can consider that costs are generated both by activities and resources. In fact, when there are several execution modes for an activity, they are often characterized by different levels of resource consumption and by different durations, inversely proportional to the resource requirements. As a consequence, we consider this type of costs as mixed.

This chapter is organized as follows. The next section provides several definitions and notations used for project scheduling problems with financial objectives. The third section presents problems for which costs are activity-related and the objective is the maximisation of project benefits. Problems with resource-related costs are studied in the last section.

14.2. Definitions and notations

14.2.1. *Definitions*

The objective of the contractor and the client is to reduce costs and ideally to maximize their profit. To maximize project profit is equivalent to maximize its Net Present Value (NPV). The NPV is the most frequent criterion used in project scheduling when the project financial aspects are taken into account. This criterion is calculated with cash flows generated by project activities. A cash flow can be positive (cash inflow) or negative (cash outflow). For the contractor, cash outflows represent the expenses caused by manpower, equipment and raw materials while cash inflows represent the client's payments (they are often proportional to the project's advancement).

The NPV was first introduced by Russell in 1970 ([RUS 70]). It is a non-regular objective function, except in the case where cash flows are only positive. NPV represents profit in actual value that will be earn after x years. NPV is defined as below (when project representation is a AOA (Activity-On-Arc[1]) graph where cash flows are associated with the n events or n graph vertices):

NPV=$\sum_{i=1}^{n} F_i e^{-\alpha T_i}$ subject to: $S_k + p_k < C_k$ with data:

F_i cash-flow generated by event i,

S_k (respectively C_k) the event associated with the start (respectively the end) of the activity A_k and p_k the duration of the activity A_k.

1. Contrarily to the Activity-On-Node representation of the project presented in Chapter 1, the Activity-On-Arc representation associates the activities with the arcs of the precedence graph, while the nodes represent events (i.e. activity start and completion times).

$e^{-\alpha} = (1+r)^{-1} = \beta$, the discount factor with r the discount rate, i.e. what we are going to lose or earn by investing some money in the project instead of keeping it.

S_i and C_i, starting time and completion time of A_i.

The discount factor is used to take the discount (i.e. the money variation in time) into account. In fact, discounting consists in giving to different cash flows occurring at different dates an actual value. A major difficulty consists in determining the adequate discount factor.

14.2.2. *Notations*

From all existing scheduling problems notations, we use the notation proposed in [HER 98b] that uses the traditional notation of traditional production scheduling problems. The α field describes resource characteristics, the β field specifies the activity characteristics and the γ field specifies the objective function. As β_8 is used to describe the financial implications of the project activities, we only detail its possible values:

– c_j if activities are associated with cash flows;

– c_j^+ if cash flows associated with activities are positive;

– $\tilde{c_j}$ if cash flows are stochastic;

– per if cash flows are periodic;

– $sched$ if both the amount and the timing of cash flows have to be determined;

– $\emptyset$ if no cash flows are specified.

In this chapter we use this notation to classify all of the studied problems. For example, the previous model can be noted $cpm, c_j|NPV$: only one cash flow is associated with a graph node (c_j) and resources are unconstrained (cpm).

To better fit the reality and to avoid ambiguity, some precisions about cash flow occurrence must be made. First of all, cash flows have to be associated with activities instead of events. In fact, associating cash flows with events generates confusions because an event can correspond with the end of several activities and the beginning of others without adaptation of the classical AOA representation of projects (adaptations were presented in [PAD 97], [YAN 95] and [DAY 96]).

Several cash flows can be associated with one activity. When an activity is quite long, discounting cash flows on one unique period can be inappropriate. A solution may consist in decomposing activities into subactivities to have only one cash flow associated with an activity, but this decomposition increases the graph complexity.

As a consequence, some authors adopt another representation, allowing several cash flows associated with one activity. This model was first introduced in [DOE 77].

It supposes there are several cash flows associated with one activity and they are imputed all together at the end of the activity. Time periods are defined by associating a unique cash flow to a time period. This model also considers the loss generated by the activity processing called *opportunity costs*. They also take capital constraints into account. It can be noted that Russell's approach [RUS 70] is a particular case of the approach of Doersch and Patterson [DOE 77].

Another remark can yet be formulated: the last two models suppose that a cash flow generated by an activity is constant, i.e. its value does not depend on the starting time of the activity. In [ETG 96] a new model is introduced where cash flows values are time dependent. This model considers that amount of cash flows is a non-increasing linear time function. Vanhoucke *et al.* [VAN 01b] studied a similar problem and introduced another value for β_8 : c_j^{lin}.

14.2.3. *Classification*

The two next sections present a review of different models found in other works. The presentation follows this classification:

– MAXNPV: NPV of the project has to be minimized, no resource constraints are taken into account;

– RCPSPDC (Resource Constrained Project Scheduling Problem with Discounted Cash Flows) corresponds to the MAXNPV with resource constraints;

– CCPSP (Capital Constrained Project Scheduling Problem) is a RCPSPDC generalization with additional constraints on capital;

– PSP (Payment Scheduling Problem): amount and occurrence of cash flows have to be determined to maximize the project's NPV.

When costs are generated by activities through their execution mode, we are faced with the TCTP (Time/Cost Trade-off Problem). When costs are resource-related we can distinguish three problems:

– RLP (Resource Leveling Problem): Resource consumption has to be leveled from one time period to another, or on each period the difference between resource usage and a target value has to be minimized;

– RIP (Resource Investment Problem) or RACP (Resource Availability Costs Problem) consists of minimizing the capacity of bought resources while respecting the requirement of activities;

– RRP (Resource Renting Problem) consists of minimizing the capacity of rented or bought resources while respecting activities requirements. Two different types of resource costs are considered. RIP is a subproblem of RRP (Resource Renting Problem).

We refer to the classical three field notation of Herroelen *et al.* [HER 98b] for further details. We recall basic notation that are used in this part. In field one (α-field) $1, 1$ means that only one renewable resource is considered and $m, 1$, means that m renewable resources are taken into account simultaneously. In field two (β-field) cpm means that activity are submitted to traditional start-end precedence relation, gpr means that generalized precedence relation are considered. Notice that in most of the studied problems, whereas the objective function is related to costs, the project deadline is fixed, which is denoted by δ_n in the second filed of the traditional notation.

Notice that related to these problems TCTP (Time/Cost Tradeoff Problem) has been the subject of many papers. For the TCTP, activities have several execution modes, each mode corresponding to a pair: duration and cost (discrete TCTP). Each activity can also have its cost and duration linked by a continuous function (continuous TCTP). In this case, an upper and a lower bounds are specified for cost and duration. The objective of TCTP can be either to minimize total costs generated by crashing some activities, either to minimise the project duration while respecting an upper bound on total project costs or to determine the whole Pareto optima curve. The TCTP complexity depends on the nature of the cost-duration relationship. Related to problems dealing with costs withouts cash flows, De Dunne, Ghosh and Wells [DE 95] gave a state-of-the-art review on TCTP. To solve discrete TCTP, the project graph can be simplified by suppressing some activities execution modes or by changing the graph structure ([BRU 83], [SCH 80], [MUL 89], [BEI 92]). The majority of approaches dealing with TCTP do not consider constraints on resources (except on the capital available for the project), and do not consider that cash flows can occur.

14.3. NPV maximization

In this section we focus on minimization of NPV of the project, without resource constraints (MAXNPV) denoted $cpm, c_j|NPV$, and then adding resource constraints (RCPSPDC), denoted $m, 1|cpm, c_j|NPV$.

14.3.1. *Unconstrained resources: MAXNPV and PSP*

MAXNPV is a polynomial problem that was first studied by Russell in 1970 [RUS 70] with a model where cash flows are associated with events of an AOA project graph ($cpm, c_j|NPV$). The proposed resolution is based on the transformation of the non-linear mathematical model of $MAXNPV$ into a linear one with the help of Taylor series approximation. This transformation results in a linear program whose dual is a transshipment network flow problem. Russell based his resolution on the Ford and Fulkerson algorithm (out-of-kilter) and found optimal solutions.

Grinold in [GRI 72] took the same model $cpm, c_j|NPV$ and added a project deadline denoted by δ_n and showed that Russell's non linear program can be linearized

directly with the use of particular structure of this non linear program. He provided two procedures, one with a fixed project deadline and the other takes the project deadline as a parameter and shows relationship between project NPV and project deadline.

Elmaghraby and Herroelen [ELM 90] proposed a new approach to solve $cpm, \delta_n, c_j|NPV$ and showed that the approaches of Grinold and Russell were imperfect. In fact, in special cases, Russell and Grinold procedures can be deadlocked or can not provide a coherent solution. Moreover, Elmaghraby and Herroelen point out the complexity of Grinold and Russell approaches and propose an algorithm based on a simpler concept: advancing as much as possible events associated with positive cash flows while delaying as much as possible those associated with negative cash flows. In [SEP 94], Sepil pointed out a flaw in Elmaghraby and Herroelen approach, corrected it and gave a computational evaluation of the modified procedure.

Schwindt and Zimmermann in [SCH 01] consider the unconstrained MAXNPV problem with general temporal constraint. The method that they proposed is a steepest ascent approach. Note that the efficiency of this algorithm is demonstrated on the basis of two randomly generated benchmark problem sets consisting of instances with up to 1000 activities.

Demeulemeester, Herroelen and Van Dommelen [DEM 96b] proposed an optimal resolution of $cpm, \delta_n, c_j|NPV$ with an AON project representation where cash flows are associated with completion time of some activities. This procedure was adapted by De Reyck and Herroelen [de 96b] to be used on $gpr, \delta_n, c_j|NPV$. The main difference between the two approaches is the necessary transformation of the generalized precedence relations into standard precedence relations (with transformation rules defined in [BAR 88]). Related to the same problem, Vanhoucke, Demeulemeester and Herroelen [VAN 03] present a branch-and-bound algorithm which computes upper bounds by making piecewise linear overestimations. The problem is relaxed into a weighted earliness-tardiness project scheduling problem.

$cpm, (\delta_n), c_j^{lin}|NPV$ was studied by Etgar, Shtub and LeBlanc [ETG 96] (simulated annealing), Shtub and Etgar ([SHT 97] (branch-and-bound procedure) and [VAN 01b] (polynomial algorithm for the $cpm, \delta_n, c_j^{lin}|NPV$ based on an implicit enumeration and on the algorithm developed by Demeulemeester, Herroelen and Van Dommelen [DEM 96b]).

Few stochastic models were developed and the stochastic characteristic is related to activities duration rather than cash flows (Buss and Rosenblatt [BUS 93], Elmaghraby *et al*. [ELM 00]).

In MAXNPV, the amount and timing of cash flows are known. However, the amount and timing of a client's payments directly affect project benefits for both the client and the contractor. The amount and timing of progress payments are one of

the most important points in the negotiations between client and contractor. In fact, the contractor's ideal solution is to perceive one unique contractual payment at the beginning of the project. After receiving this payment, the contractor attempts to minimize costs by scheduling activities associated with a large investment as late as possible. Of course, this is not the client's ideal solution. The client prefers to give one unique payment at the end of the project. Thus, for the client, the schedule of activities is less important than the completion time of the project.

The objective of negotiations between the contractor and the client is to obtain an equitable solution. An equitable solution is defined as a solution in which client and contractor deviate from their ideal solutions with a same percentage.

PSP (Payment Scheduling Problem) can be studied for the contractor point of view as for the client's one. In this problem, the amount and occurrence of cash flows have to be determined to maximize the project NPV.

Dayanand and Padman, studied PSP ($cpm, sched|NPV$) from the contractor's point of view. In this problem both the amount and timing of cash flows have to be determined (denoted by $sched$) In [DAY 93], they gave an integer linear program and tested several heuristics.

14.3.2. *Constrained Resources: RCPSPDC, CCPSP and PSP*

Here we consider NPV minimization subjected to resource constraints: $m, 1|cpm, c_j|NPV$.

The RCPSPDC and CCPSP problems are NP-hard as RCPSP generalizations. The majority of procedures developed to solve this problem are heuristics but there exist some exact methods that are applicable on small size instances. The mathematical model of this problem is similar to the MAXNPV model with supplementary constraints on the resource capacities (see Chap. 3). For this problem, as far as our knowledge, literature references only renewable resources

Yang *et al.*[YAN 92] studied $m, 1|cpm, \delta_n, c_j|NPV$ and assumed AOA networks in which an activity can be associated with several cash flows. Notice that δ_n denotes a project deadline. Authors add bonus and penalties for lateness or advance are associated with the project completion time. They used an implicit enumeration algorithm developed by Talbot and Patterson [TAL 78] and gave two dominance rules to avoid enumerating all possible solutions. This procedure was evaluated on 20 problems and the authors concluded that it can fail to find an optimal solution for problems made up of more than 30 activities or for specific problems (with tight resource constraints, a deadline higher than the project optimal duration for the makespan minimisation).

Icmeli and Erenguc [ICM 96a] developed a branch-and-bound procedure for the case where the resources may be available in variable amount (denoted by va):

$m, 1, va|cpm, \delta_n, c_j|NPV$. Nodes correspond to complete solutions compatible with precedence constraints but not with resource constraints. Initially, the lower bound is given by a heuristic procedure or set to $-\infty$. The upper bound is calculated with Grinold [GRI 72] procedure (see section 14.3.1) on the transformed problem (initial problem without resource constraints). The branching scheme is the one developed in the DH-procedure [DEM 92b]. A node is cut off if it is associated with an upper bound lower than the lower bound or if it conducts to a solution whose completion time is upper than the project deadline. When a node is associated with a feasible solution, the upper bound is updated. Tests on Patterson set (50 problems) and the PROGEN set (40 problems) show that this procedure obtains better results than the procedure given in [YAN 92] and described above. Baroum and Patterson [BAR 99] also developed a branch-and-bound procedure on the same branching scheme for $m, 1, va|cpm, c_j^+|NPV$ but nodes of the searching tree correspond to feasible partial solutions.

Vanhoucke, Demeulemeester and Herroelen [VAN 01c] introduce a depth-first branch-and-bound algorithm for which makes use of extra precedence relations to resolve a number of resource conflicts. Upper bound for the MAXNPV problem is also proposed in this case based on the idea that that positive cash flows should be scheduled as early as possible while negative cash flows should be scheduled as late as possible within the precedence constraints.

Several heuristics were also proposed to solve the RCPSPDC, for example: Icmeli and Erenguc [ICM 94] used a Tabu search procedure to solve $m, 1|cpm, \delta_n, c_j|NPV$ and Yang *et al.* [YAN 95] proposed procedure based on stochastic rules. Recent developments on heuristic method for this problem have been proposed by Vanhoucke [VAN 06d], based on recursive procedure that can also be used as a part of more global optimization method.

De Reyck and Herroelen [de 98] developed a branch-and-bound method for the $m, 1, va|gpr, \rho_j, \delta_j, vr, c_j|NPV$, ($\rho_j$, means that ready times differ per activity and vr that the resource requirements may be variable discrete). Each node of the searching tree corresponds to a complete solution and the branching scheme is extracted from the DH-Procedure [DEM 97]. They also proposed three dominance rules to avoid the exploration of all the nodes of the searching tree. This branch-and-bound method solves problems with up to 50 activities and its truncated version gives also good results.

Suboptimal approaches are numerous to solve $m, 1|cpm, c_j|NPV$. Russell [RUS 86] was the first to test standard priority rules for this problem. Several other authors developed procedures based on priority rules. Zhu and Padman [ZHU 97] used both neural networks and priority rules to solve this problem. Recently Mika *et al.*[MIK 05], considered metaheuristics (simulated annealing and tabu search) for solving MM-RCPSP with discounted cash flows. They investigate four different

payment models: lump-sum payment at the completion of the project, payments at activities' completion times, payments at equal time intervals and progress payments.

Periodic cash flows (denoted by per) were studied in [SEP 97] in which the authors adapted the model developed by Kazaz and Sepil [KAZ 96] for $cpm, per|NPV$ to take resource constraints into account and to solve $m, 1|cpm, per|NPV$.

Buss and Rosenblatt [BUS 95] and Elmaghraby [ELM 00] present a model for the NPV maximization when activities duration are stochastic.

The CCPSP is a RCPSPDC generalization. It differs from the RCPSPDC by considering constraints capital available for the projet.

Doersch and Patterson [DOE 77] introduced a binary integer linear programming formulation to solve $1, 1, va|cpm, \delta_n, c_j|NPV$. This model solves to optimality projects with 15 to 20 activities but fails to solve problems with more than 30 activities. This approach was updated in [SMI 96].

Smith-Daniels and Smith-Daniels [SMI 87] extended the Doersch and Patterson model to take constraints on cost and on material into account: $m, 1, va|cpm, \delta_n, c_j|NPV$. This procedure allows solving small problems to optimality.

For the resource-constrained PSP with multiple modes of activity processing (denoted by mu), Ulusoy and Cebelli [ULU 00] proposed a double-loop genetic algorithm to solve $m, 1|cpm, \delta_n, mu, sched|NPV$. The objective is to determine an equitable solution for both the contractor and the client.

Some hybrid approaches can be found that combine TCTP with cash flows on activities and discounting with constraints on resources: [TAL 82], [ERE 93], [ICM 96b]. Other extensions are considered in [VAN 08b] : time/switch constraints, work continuity constraints and net present value maximization. Authors present both an exact solution approach for the work continuity version and a new meta-heuristic approach.

14.4. Resource-related costs

In the first section of this chapter we have presented a classification and a review of the researches considering scheduling problems where costs are generated by activities through their execution mode: TCTP (Time/Cost Trade-off Problem). In this section we assume that costs are resource-related then we can distinguish two main classes of problems: costs can be caused by sub-optimal use of the resource (RLP) or the necessary reservation of resources (RIP).

14.4.1. *Resource leveling problem*

This problem consists of minimizing the maximal resource utilization by avoiding peaks in resource consumption either by limiting deviations between the resource consumption and a target value or by minimizing variations of resource utilization over time. The complexity of RLP depends if the resources are constrained or not.

The objective function can consist in:

– minimizing the absolute value of the difference between resource usage and a target value, $\sum abs.dev$;

– minimizing the square of the difference between resource usage and a target value, $\sum sq.dev$;

– minimizing jumps in resource usage between two consecutive time periods, $\sum jump$;

– minimizing the maximal resource usage, $maxreq$.

Ahuja [AHU 76], Easa [EAS 89], Bandelloni *et al.* [BAN 94] and Younis and Saad [YOU 96] proposed exact algorithms based on implicit enumeration, on integer programming or on dynamic programming. Easa devised a procedure for $1,1|cpm,\delta_n|\sum abs.dev$ and $1,1|cpm,\delta_n|\sum jump$ (δ_n means that the project deadline is fixed). His procedure finds the optimal solution for small to medium-size projects. Bandelloni's dynamic programming approach solves $1|gpr,\delta_n|\sum abs.dev$ and leads to an optimal solution for small and medium projects.

Savin ([SAV 96], [SAV 97]) used neural networks to solve the RLP.

Heuristic approaches were proposed by several authors, all of them consist of moving some activities or in computing a priority for each activity.

Harris [HAR 90] showed that when a resource is leveled, and if the resource demand are uniform for all resources, other resources tend also to be leveled. He developed a heuristic procedure to level the utilization of a unique resource, with generalized precedence relations. His procedure is easy to apply and is applicable on some $1,1|gpr,\delta_n|\sum jump$ problems. The first step of his procedure consists of building the base resource usage histogram by considering only critical activities. Others activities are placed with the help of three priority rules. An example is detailed but no test was carried out.

For RLP with generalized precedence relations and general objective function, Brinkmann and Neumann [BRI 96] devised two greedy algorithms to solve $m,1|gpr,\delta_n|Cmax$ and $m,1|gpr,\delta_n|\gamma$ (with $\gamma = \sum jump$ or $= maxreq$). Their procedure is based on implicit enumeration or on graph reduction (contraction).

Tests were only carried on $m|gpr, \delta_n|\gamma$ with $\gamma = \sum jump$ or $\gamma = maxreq$ with unconstrained resources.

Neumann and Zimmermann [NEU 99] devised three polynomial heuristics for $m|gpr, \delta_n|\gamma$ or $m, 1|gpr, \delta_n|\gamma$ with $\gamma = \sum abs.dev$, $\sum sq.dev$, $\sum jump$ or $maxreq$. Problems composed of up to 500 activities and 5 resources can be efficiently solved.

The same authors (in [NEU 00b]) have developed a branch-and-bound method (for RLP with or without constrained resources) and a heuristic procedure, starting from a feasible solution of resource-unconstrained RLP and enhance it to obtain a feasible solution to the resource-constrained RLP. Solved problems are $m|gpr, \delta_n|\gamma$ with $\gamma = \sum abs.dev$, $\sum sq.dev$, $\sum jump$ or $maxreq$. Neumann and Zimmermann also developed an adaptation of their BB procedure with the filtered beam search method.

14.4.2. *Resource Investment Problem (RIP) and Resource Renting Problem (RRP)*

In the Resource Investment Problem (RIP), also denoted by the Resource Availability Cost Problem (RACP), costs are generated by the reservation of a given amount of resource for the project. A solution of this problem consists of a set of activities starting time and a set of resource capacities, while respecting a project deadline. This problem is $\mathcal{NP}$-hard

Some authors argue that RIP is a Resource Leveling Problem RLP subproblem. If the two problems are strongly related in RLP, resource capacities are known while they are variables and have to be determined in RIP.

In RIP the system has bought all the resources needed by the project. The cost of a resource is proportional to the resource capacity, is not time-dependent and does not vary if the resource is used once or during all the project. In RRP, resources can be rent and costs generated by a resource are of two types: fixed costs to have the resource R_k : c_k^p , that are not time-dependent, and renting costs, c_k^r. Though, disposing of the resource R_k during t time periods costs $c_k^p + t \cdot c_k^r$. RRP is $\mathcal{NP}$-hard as a RIP generalization.

The objective of RRP consists of minimizing resource total costs while respecting activities requirements and deadline of the project. All the resources needed by the project are known and there are renewable.

RRP is denoted by $m, 1|cpm, \delta_n|rrc$ (rrc = *resource renting costs*). A RRP solution is a 3-tuple composed by the activity starting times, the number of resource units added at each time period (α_{kt}) and the number of resource *k* units removed at each time period (ω_{kt}).

Möhring [MÖH 84] was the first to define the RIP and to propose an algorithm for $m, 1|cpm, \delta_n|rac$ (rac=*resource availability costs*). Möring's approach is based on the Radermacher [RAD 85] procedure for the RCPSP.

Demeulemeester [DEM 95] devised a procedure to solve the same problem, based on a branch-and-bound procedure for the RCPSP developed by Demeulemeester [DEM 92a] (see Chapter 5). Two different strategies are presented, the first is based on the progressive increase of resource capacity and the second is based on the progressive decrease of resource capacities. The first strategy can not guarantee the optimality of the solution but a feasible solution can be obtained quickly. The second strategy is based on the decision problem associated with RCPSP : the RCPSDP, solved for an adaptation of the initial problem. This adaptation considers only two constrained resources, other resources capacities are supposed to be infinite. The minimal resources capacities can be computed. This resolution allows the construction of non-strict Pareto optima set. Within this set, the Pareto optimum associated to the lowest cost is chosen and the RCPSDP associated to this vector is solved. If no feasible solution exists, the optima Pareto set is updated with new points. This strategy guarantees that the first feasible solution found is an optimal one. If the decision problem included in [DEM 95] procedure is changed, other RIP problems can be solved by this procedure : RIP with general precedence constraints on activities or RIP with activity preemption. The method proposed in [DEM 95] obtains better results than the method proposed in [MÖH 84] because the procedure to solve the RCPSP developed in [DEM 92a] outperforms the procedure developed in [RAD 85].

Two approaches were proposed by Kimms in [KIM 01], for $m, 1|cpm, \delta_n|rac$, one based on Lagrangean relaxation, the other one on column generation. Computational tests show that Lagrangean relaxation obtains worse results than column generation but needs less computational time. For 64% of the instances, the Lagrangean relaxation leads to solutions close to optimal solution (less than 10 %).

The RIP can be used to solve staff management problems. In fact, the problem of the workforce size determination can be modeled as a RIP. For example, Kane and Baptiste [KAN 01] solved $3, 1, va|cpm, \delta_n|rac$. They aim at determining the optimal numbers of interim workers and of supplementary hours to minimise salary costs while respecting workload. The objective function takes the French legislation on maximal weekly working hours into account by allowing the definition of two different supplementary hours' prices.

For the RRP, Nübel in [NÜB 01] developed an exact procedure to determine an optimal renting policy, knowing a feasible schedule of project activities. He also gave a "depth-first" branch-and-bound method that enumerates all quasi-stable schedules and determines an optimal renting location (Nübel shows that the set of quasi-stable schedules is dominant). Each node of the searching tree corresponds to a partial solution. If at a node p all the activities starting times are fixed, an optimal renting policy

is computed, the upper bound is updated and the backtrack occurs. Otherwise, nodes are created by fixing a new activity. Authors also gave three lower bounds. They test their procedure on 10,800 problems form ProGen/max, showing that their branch-and-bound is efficient on small size instances (up to 10 activities). They concluded that the RRP difficulty is proportional to the ratio c_k^r/c_k^p. A branch-and-bound method for the RRP is also given in [NEU 03].

14.5. Conclusion

Figure 14.5 presents a synthesis of the studied problems. For further reading, and eventually missing references we refer to Drezet (2002). This figure highlights the heterogeneity of the interest researchers have on the financial aspect in project scheduling.

Nevertheless, the vast majority of these approaches remains quite far from the preoccupation of the project managers in industries. None of the proposed models are available in standard project scheduling software. The gap between theoretical and pragmatic solution is still present. It is probably due to the complexity to determine the right discount rate or to estimate the requirements of resources or costs generated by the booking of these resources.

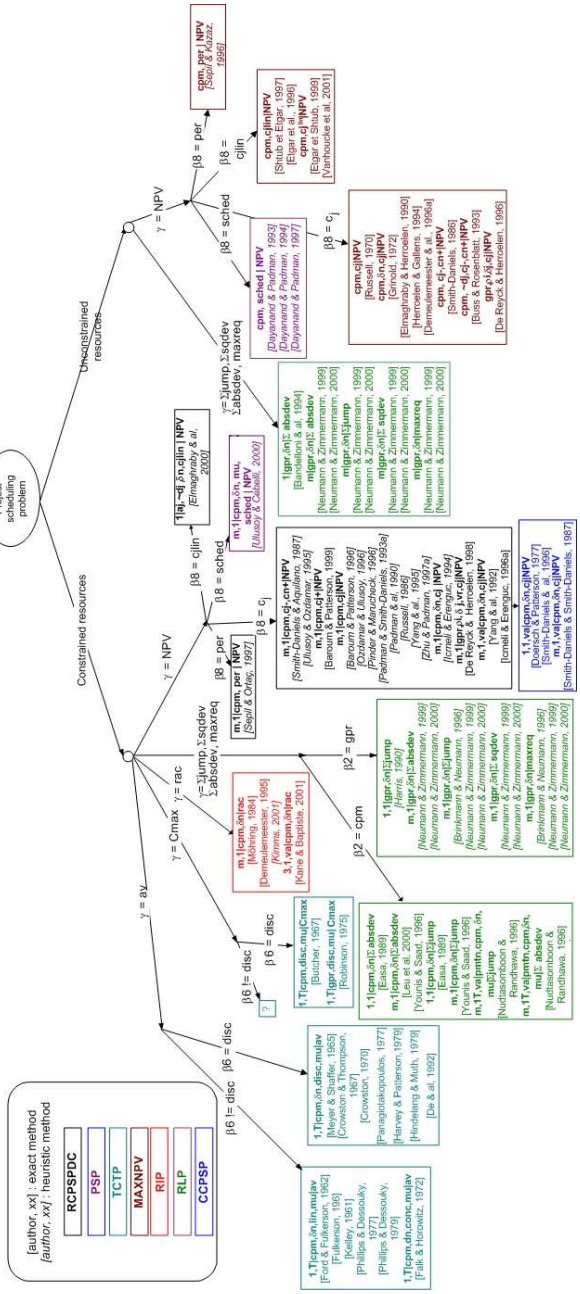


Figure 14.1. *A summary of approaches for RCPSP with financial costs*

THIRD PART

Industrial Applications

Chapter 15

Assembly Shop Scheduling

The objective of this chapter is to present an industrial problem resulting from a collaboration with the French automotive company PSA and to give two extensions of the RCPSP which make it possible to take into account some constraints of the industrial problem. For each extension, a mathematical model is presented. Dedicated methods and some results are also given.

15.1. The assembly shop scheduling problem

The assembly shop scheduling problem has first been studied with one shop (mono shop problem) before been studied on the whole (multi shop problem). We present the mono shop problem and the multi shop problem by using the decomposition in three subsystems [GOU 91]:

– the **physical subsystem** is composed of the physical entities of the system. It makes it possible to describe the topology of the workshop, the technical characteristics of the production resources and the physical and logical links between the resources;

– the **logical subsystem** is composed of components to manufacture and the associated set of manufacturing routing. It describes the steps to manufacture the products in terms of raw materials, semi-finished products, processing times,etc.;

– the **decisional subsystem** or management subsystem specifies the functioning rules of the system and the algorithms to pilot the entities. It is composed of a set

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

of management or dispatching rules which act and modify the logical and physical subsystems.

15.1.1. *Mono shop problem*

15.1.1.1. *Physical subsystem*

An assembly shop (Figure 15.1) is divided into elementary surfaces (two in the width of the shop and m in the length) and is dedicated to the assembly of spare parts. The assembly of the parts requires human resources (fitters and operators) and material resources: different types of welding pliers, identical movable rails used to carry the pliers and assigned either to the *high area* of the shop or to the *low area*. All the resources are available in a limited number in a given shop.

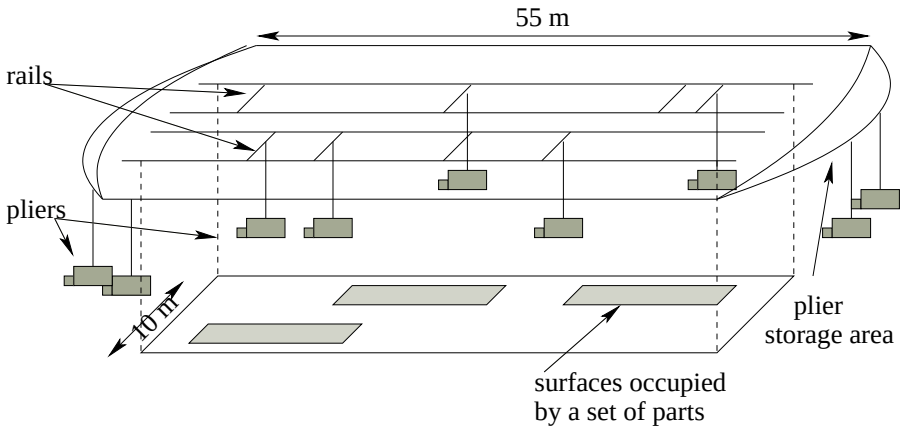


Figure 15.1. *Representation of an assembly shop*

15.1.1.2. *Logical subsystem*

The logical subsystem is composed of sets of parts. All the parts, from a given set are assembled according to the same treatment. The processing is subject to precedence constraints (nomenclature). The assembly of a set requires human and material resources. The assembly of a set is divided into three steps as shown by Figure 15.2: installation of the set in the shop (done by the fitters), assembly of the set (done by the operators), and removal of the set from the shop (done by the fitters).

Sets of parts must be placed on one or more surfaces according to their size. Each set of parts can be placed in the shop according to, at most, two orientations (Figure 15.3). The number of contiguous elementary surfaces required in the width and in the length of the shop and the material resources (pliers, rails) required depends on the chosen orientation.

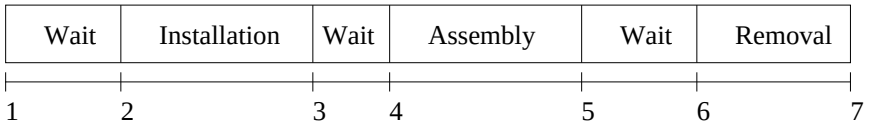
A set of parts is characterized by:

– the number of required human resources,

– the duration of the installation and removal steps,

– the number of possible orientations (at most 2),

– for each orientation:
  - the number of elementary surfaces, required in the width and in the length,
  - the number of required material resources,
  - the duration of the assembly step.



1) Ask for required resources: fitters, surfaces, pliers and rails

2) Assignment of required resources; beginning of the installation of the set of parts

3) End of the installation of the set of parts; release of the fitters; ask for operators

4) Assignment of operators; beginning of the assembly step

5) End of the assembly step; release of the operators; ask for fitters

6) Assignment of the fitters; beginning of the removal of the set of parts

7) End of the removal; release of the surfaces, rails, pliers and fitters

Figure 15.2. *Events relatives to the treatment of a set*

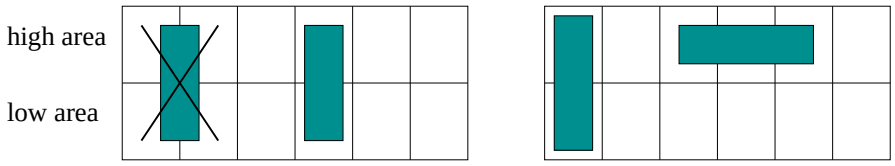


Figure 15.3. *Examples of orientations*

15.1.1.3. *Decisional subsystem*

The decisional subsystem is composed of:

– a **scheduling problem**: the sets of parts, in input of the shop, have to be scheduled;

– an **assignment problem**: the human and material resources have to be assigned to the sets before the assembly and the sets of parts have to be laid in the shop.

The proposed solution must respect the resource availability constraints, the precedence constraints and the constraint about the contiguity of the surfaces. The objective is to minimize the makespan.

More precisely, two problems have been identified:

– **Problem 1**: mono shop problem with an unlimited number of material and human resources. The objective is to find a schedule of the sets and an assignment of the surfaces to the sets which respect the precedence constraints, the constraint about the contiguity of the surfaces and to minimize the makespan. Solving this problem makes it possible to estimate the performance of the shop with only the limitation of the number of available elementary surfaces.

– **Problem 2**: mono shop problem with a limited number of material and human resources. The objective is to find a schedule of the sets and an assignment of the resources to the sets which respect the availability and the precedence constraints, the constraint about the contiguity of the surfaces and to minimize the makespan. This problem helps to study the impact of the limited number of resources. Solving this problem makes it possible to evaluate the opportunity to increase or to decrease the number of resources of each type.

15.1.2. *Multi shop problem*

The physical subsystem is composed of many shops. The resources of each shop, called local resources, are completed with global resources, which can be used in any shop. Global resources are pliers, fitters and operators, and are also available in a limited number. The logical subsystem is the same as in the mono shop problem, but a set of parts can be assembled in any shop. The decisional subsystem is the same as in the mono shop problem, but an assignment problem of the sets of parts and of the global resources to the shops is added. The proposed solution must respect the local and global resource availability constraints and the constraint about the contiguity of the surfaces. The objective is to minimize the makespan.

More precisely, two problems have been identified:

– **Problem 3**: multi shop problem without sets distribution. The distribution of the sets in the shops is supposed to be established, the objective is to find a schedule of the sets of parts in each shop and an assignment of global and local resources to the sets which respect the availability and the precedence constraints, the constraint about the contiguity of the surfaces and to minimize the makespan.

– **Problem 4**: multi shop problem with sets distribution. This problem is the whole problem. The objective is not only to determine a schedule of the sets in each shop and an assignment of global and local resources to the sets, but also to distribute the sets among the shops in order to respect the availability and the precedence constraints, the constraint about the contiguity of the surfaces and to minimize the makespan.

The impact of global resources can be analyzed by comparing separate studies (Problem 2) for many shops with a multi shop study (Problem 3).

Figure 15.4 represents the four identified problems and Table 15.1 describes the characteristics.

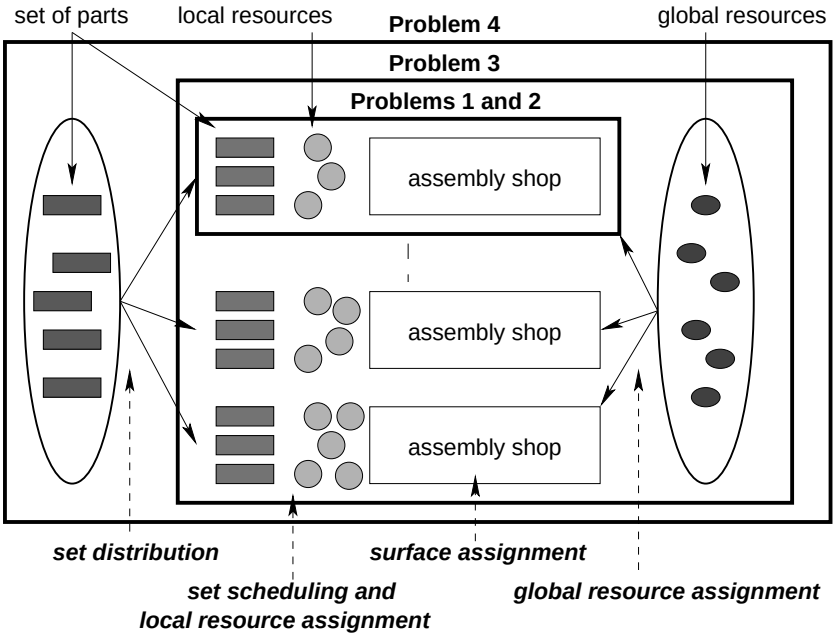


Figure 15.4. *The identified problems*

	Shop		Surfaces	Resources		Set
	mono	multi		Local	Global	distribution
Problem 1	X		X			
Problem 2	X		X	X		
Problem 3		X	X	X	X	
Problem 4		X	X	X	X	X

Table 15.1. *Characteristics of the problems*

15.2. Link with the RCPSP

The industrial problem can be assimilated to a Multi Mode-RCPSP (MM-RCPSP). Indeed, activities, to which resources have to be assigned, have to be scheduled and the number of available resources is limited. The number of required resources depends

on the orientation (mode). However, we have noticed some differences with the MM-RCPSP:

– **resource individualization**: in RCPSP models, resources of a given type are not individualized. When an activity requires resources from a given type, no distinction between resources is enabled: we only need to know if the number of available resources is sufficient. In our problems, resource individualization is required, in order to take into account the surface contiguity constraint. The assignment of elementary surfaces to a set of parts can be done only if a sufficient number of contiguous surfaces is available, according to the chosen orientation. To take into account this constraint, it is not sufficient to check that the quantity of surfaces, used during each period is lower or equal to the number of available surfaces. We need to know which surface is assigned to each set of parts.

– **variation of the number of resources required by the activities**: in RCPSP models, if an activity lasts many periods, then the number of resources required by the activity is the same for each period. In our problem, fitters and operators do not take part to the whole assembly steps. Fitters take part to the installation and removal steps, whereas operators take part to the assembly step. The duration of each step is known, but it is necessary to add to these steps two delays: delay for waiting for operators between the end of the installation step and the beginning of the assembly step and delay for waiting for fitters between the end of the assembly step and the beginning of the removal step. The duration of the delay is unknown.

Few papers consider variable resource demand profile. Li and Willis [LI 92b] deal with a RCPSP which includes renewable and non renewable resources. For the renewable resources, the demand profile is variable. The activities are divided into sections and the number of required resources depends on the sections. However, an activity must be continuously realized: no delay is allowed between two successive sections. The authors propose a forward-backward multi-pass heuristic. Poder *et al.* [POD 04] study a RCPSP in which the resource demand profile is not constant but varies according to the period. Each activity is decomposed into consecutive sub activities and the resource demand profile is modeled by a continuous function. Dauzère-Pérès *et al.* [DAU 96] deal with a scheduling problem in a multi-resource shop. A set of operations, linked by precedence constraints, must be processed by a set of resources. Each operation requires many resources and each resource can be chosen among a given set of resources. The sets of resources are not necessarily disjoint. The problem consists of assigning the operations to the resources and scheduling the operations on the resources in order to minimize the makespan. The authors propose a graph to model the problem and a neighboring system for a tabu search. This approach has been extended in [DAU 03] in order to take into account two additional constraints: incompatibilities between resources, and release of resources before the end of the processing of an operation. However, the number of required resources for a given activity can only decrease during its execution, the increase is not taken into account.

The resource individualization has been already study, in particular in case of human resources. For examples, we can refer to chapters 9 and 16.

15.3. Proposition of extensions

From an academic point of view, this industrial problem brought us to propose extensions:

– **RCPSP with variable demand profile**: an activity is divided into steps. Two consecutive steps require a different number of resources of a given type. This variation of the number of resources required during the treatment of an activity may cause waiting time between the steps (because of the possible unavailability of resources). Thus, the duration of an activity is not known in advance.

– **RCPSP with resource individualization**: this extension makes it possible to give, for each activity, not only the starting time of the activity but also one or more accurate assigned resources.

15.3.1. *RCPSP with variable demand profile*

The considered hypotheses are the following:

– there are different types of resources and they are renewable ; for each type and each period, the number of resources is limited;

– the activities are subject to precedence constraints such as: activity j can not start before the completion of activity i;

– the activities are not preemptive;

– each activity is composed of at least one step;

– for each step, the duration is known, as well as the required quantity of resource from each type (constant during the process of the step);

– two successive steps have different resource requirements at least for one type of resource;

– it is not compulsory to process continuously two successive steps. They can be separated by a waiting time due to the unavailability of a resource;

– if two successive steps require the same resources, their resources are not released during the waiting time and can not be used for another activity.

To the definitions already defined in chapter 1, we add the following ones:

– **Steps**:

E_i	number of steps in activity A_i
$p_{i,j}$	processing time of step j of activity A_i
$b_{i,j,k}$	amount of resource R_k used during the execution of step j of activity A_i
$C_{i,j}$	completion time of step j of activity A_i
$EC_{i,j}$	earliest completion time of step j of activity A_i
$LC_{i,j}$	latest completion time of step j of activity A_i

– **Variables**:

y_{ijt}	$= 1 \Leftrightarrow C_{i,j} = t$, step j of activity A_i finishes during period t.

We assume that activities A_0, and A_{n+1} are composed of only one step: $E_0 = 1$, $E_{n+1} = 1$.

The originality of this extension consists of the constraint to keep the resources between two successive steps. To illustrate, we consider two successive steps j and $j+1$ of an activity A_i and only one type of resource k,

– if $b_{i,j,k} > b_{i,j+1,k}$, then step $j+1$ starts immediately after the end of step j and the set of unused resources by step $j+1$ are released. The number of unused resources is equal to $b_{i,j,k} - b_{i,j+1,k}$.

– if $b_{i,j,k} < b_{i,j+1,k}$, then step $j+1$ can start only when the additional resources required for the execution of step $j+1$ are available. The number of additional resources is equal to $b_{i,j+1,k} - b_{i,j,k}$, but the $b_{i,j,k}$ resources used both by step j and by step $j+1$ are not available for other activities during the possible time between the end of step j and the beginning of step $j+1$.

EXAMPLE.– Table 15.2 describes an instance of a project with 5 activities and 2 types of resources: $\mathcal{A} = \{A_1, \ldots, A_5\}$, $\mathcal{R} = \{R_1, R_2\}$, and $B_1 = B_2 = 4$. For each activity i, we give the successors and the number of steps E_i, and for each step j of each activity, the duration p_{ij} and the number of required resources of each type b_{ijk}. Figure 15.5 represents an optimal solution.

Activity	Successor	#steps	Duration of each step	#resources required by step 1		step 2		step 3	
				R_1	R_2	R_1	R_2	R_1	R_2
A_1	A_5	3	{2,2,3}	2	3	1	0	3	1
A_2	A_4	3	{2,1,2}	2	1	2	3	2	0
A_3	A_5	2	{1,2}	0	2	2	2	0	0
A_4	-	3	{2,1,2}	1	1	2	0	1	3
A_5	-	3	{1,2,2}	2	2	3	2	3	1

Table 15.2. *Instance of a RCPSP with variable demand profile*

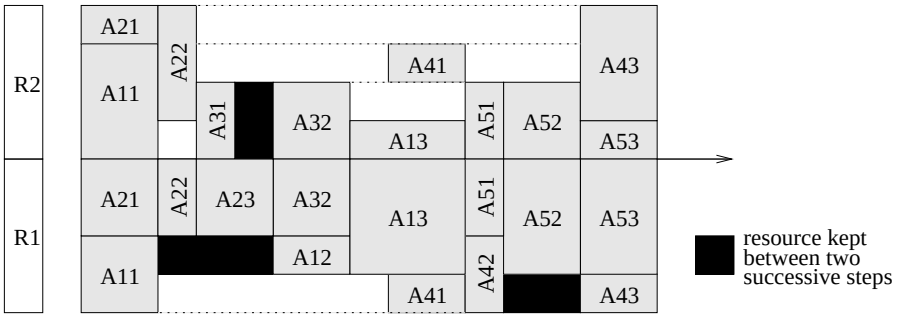


Figure 15.5. *An optimal solution*

Many objective functions can be considered. The makespan is computed by:

$$\min \sum_{t=EC_{n+1,1}}^{LC_{n+1,1}} t.y_{n+1,1,t} \tag{15.1}$$

The constraints are the following:

$$\sum_{t=EC_{i,j}}^{LC_{i,j}} y_{i,j,t} = 1 \qquad \forall\, i = 0..n+1,\ j = 1..E_i \tag{15.2}$$

$$\sum_{t=EC_{i,j}}^{LC_{i,j}} t.y_{i,j,t} + p_{i,j+1} \leq \sum_{t=EC_{i,j+1}}^{LC_{i,j+1}} t.y_{i,j+1,t} \qquad \forall\, i = 1..n,\ j = 1..E_{i-1} \tag{15.3}$$

$$\sum_{t=EC_{j,E_j}}^{LC_{j,E_j}} t.y_{j,E_j,t} + p_{i,1} \leq \sum_{t=EC_{i,1}}^{LC_{i,1}} t.y_{i,1,t} \qquad \forall\, i = 1..n+1,\ j \in \Gamma_i^{-1} \tag{15.4}$$

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{E_i-1} \min(b_{i,j,k}, b_{i,j+1,k}).(\sum_{s=t}^{H}(y_{i,j+1,s} - y_{i,j,s}) - \sum_{s=t}^{t+p_{i,j+1}-1} y_{i,j+1,s})\right)$$

$$+ \sum_{i=1}^{n}\sum_{j=1}^{E_i} b_{i,j,k} \sum_{s=t}^{t+p_{i,j}-1} y_{i,j,s} \leq B_k \qquad \forall\, k = 1..q,\ t = 0..H \tag{15.5}$$

$$y_{i,j,t} \in \{0,1\} \qquad \forall\, i = 0..n+1,\ t = 0..H,\ j = 1..E_i \tag{15.6}$$

Constraints (15.2) assure that each step of each activity is completed at only one period. Constraints (15.3) and (15.4) concern the precedence constraints. Constraint (15.3) ensures that, for each activity, step $j+1$ follows step j. Constraint (15.4) ensures that if an activity j is a predecessor of an activity i, then the time between the completion of the last step of activity j and the completion of the first step of activity i is at least equal to the duration of the first step of activity i. Constraints (15.5) are constraints on the availability of renewable resources They ensure that for each period and each type of resource, the number of used resources (by each step in process or kept between two consecutive steps) is lower or equal to the number of available resources.

15.3.2. *RCPSP with resource individualization*

The considered hypotheses are the following:

– there are different types of resources and they are renewable; for each type and each period, the number of resources is limited;

– the activities are subject to precedence constraints such as: activity j can not start before the completion of activity i;

– the activities are not preemptive;

– the duration of each activity is known, as well as the required quantity of resource from each type (constant during the process of the activity);

– the availability periods of each resource are known;

– the resources used by an activity must be compatible with the activity and between them.

To the already defined notations, we add the following ones:

– **Resources**:

$\sigma_{k,u,t}$ $= 1 \Leftrightarrow$ resource u from R_k is available during period t

$\lambda r_{k_1,u_1,k_2,u_2}$ $= 1 \Leftrightarrow$ resource u_1 from R_{k_1} is compatible with resource u_2 from R_{k_2}

$\lambda a_{i,k,u}$ $= 1 \Leftrightarrow$ activity A_i is compatible with resource u from R_k

– **Variables**:

$\omega_{i,k,u}$ $= 1 \Leftrightarrow$ resource u from R_k is assigned to activity A_i

$y'_{i,t}$ $= 1 \Leftrightarrow C_i = t$, activity A_i finishes during period t

Many objective functions can be considered. The makespan is computed by:

$$\min \sum_{t=EC_{n+1}}^{LC_{n+1}} t.y'_{n+1,t} \tag{15.7}$$

The constraints are the following:

$$\sum_{t=EC_i}^{LC_i} y'_{i,t} = 1 \qquad \forall\, i = 0..n+1 \tag{15.8}$$

$$\sum_{t=EC_j}^{LC_j} t.y'_{j,t} + p_i \leq \sum_{t=EC_i}^{LC_i} t.y'_{i,t} \qquad \forall\, i = 0..n+1,\ j \in \Gamma_i^{-1} \tag{15.9}$$

$$\sum_{u=1}^{B_k} \omega_{i,k,u} = b_{i,k} \qquad \forall\, i = 1..n,\ k = 1..q \tag{15.10}$$

$$\omega_{i_1,k,u} + \omega_{i_2,k,u} + \sum_{s=t}^{t+p_{i_1}-1} y'_{i_1,s} + \sum_{t_2=t}^{t+p_{i_2}-1} y'_{i_2,t_2} \leq 3 \qquad \forall\, i_1, i_2 = 1..n \text{ with } i_1 < i_2,$$

$$k = 1..q,\ u = 1..B_k,\ t = 0..H \tag{15.11}$$

$$\sum_{i=1}^{k} b_{i,k}. \sum_{s=t}^{t+p_i-1} y'_{i,t} \leq B_k \qquad \forall\, t = 0..H,\ k = 1..q \tag{15.12}$$

$$\omega_{i,k,u} = 0 \qquad \forall i = 1..n,\ k = 1..q,\ u = 1..B_k \text{ with } \lambda a_{i,k,u} = 0 \tag{15.13}$$

$$\omega_{i,k_1,u_1} + \omega_{i,k_2,u_2} \leq 1 \qquad \forall i = 1..n,\ k_1, k_2 = 1..q \text{ with } k_1 \leq k_2,\ u_1 = 1..B_{k_1},\ u_2 = 1..B_{k_2} \text{ with } \lambda r_{k_1,u_1,k_2,u_2} = 0 \tag{15.14}$$

$$\sum_{s=t}^{t+p_i-1} y_{i,t} + \omega_{i,k,u} \leq 1 \qquad \forall\, i = 1..n,\ k = 1..q,\ u = 1..B_k,\ t = 1..H \text{ with } \sigma_{k,u,t} = 0 \tag{15.15}$$

$$y_{i,t} \in \{0,1\} \qquad \forall\, i = 0..n+1,\ t = 0..H \tag{15.16}$$

$$\omega_{i,k,u} \in \{0,1\} \qquad \forall\, i = 0..n+1,\ k = 1..q,\ u = 1..B_k \tag{15.17}$$

Constraints (15.8) are constraints on the processing of the activities; they assure that each activity finishes during only one period. Constraints (15.9) concern the precedence constraints. Constraints (15.10) and (15.11) are constraints on the renewable resource available in a limited number during each period. Constraints (15.10) ensure that for each activity, the required quantity of resources is respected for all types of resources. Constraints (15.11) ensure that a resource is used by only one activity at a given time. Constraints (15.12) are constraints about the availability of

resources. Constraints (15.13) are incompatibility constraints between activities and resources. Constraints (15.14) are resource incompatibility constraints. Constraints (15.15) are constraints about the non-available resources.

15.4. Proposition of solution methods

The studied assembly shop scheduling problem integrates the extension about the variable demand profile and the extension about the resource individualization. Note that the second extension does not take into account the surfaces.Then, it becomes necessary to carry on the surface contiguity. Moreover, the multi-mode aspect has to be carried out. The used methods are mainly metaheuristics [BOU 01]. We have first studied the mono shop problem then the multi shop problem. In both cases, a simulation model has been built in order to compute the makespan [BEA 01].

A solution x is characterized by the concatenation of three vectors:

$$x = \sigma \& S \& M$$

where σ_i represents the i^{th} set of parts to begin its process, S_i represents the surface used by the top left corner of set σ_i and M_i is the used orientation for set σ_i. The used neighboring systems are traditional neighboring systems for scheduling (insertion of a randomly chosen set into a randomly chosen position, exchange of two randomly chosen sets) and for assignment (random choice of one or more sets and modification of assigned surfaces). One of the chosen neighboring system, noted $V(\sigma, S, M)$ consists of exchanging two randomly chosen sets and modifying the orientations and the surfaces assigned to these sets. The metaheuristics using this kind of neighboring system are combined with a simulation model for the evaluation of the criterion.

We have also used another solution representation restricted to the scheduling: $\sigma = (\sigma_1, \sigma_2, ..., \sigma_N)$ where σ_i represents the i^{th} set of parts to begin its process. A priority based heuristic for the assignment has been defined. Each time the assignment of a set is considered, the heuristic determines the orientation and the assigned surfaces. The orientations are considered according to the increasing number of used surfaces. The surfaces are considered according to three policies:

– **H1**: surfaces located on the high area of the shop are first considered (if there are many possibilities, H2 is used);

– **H2**: surfaces located on the left of the shop are first considered (if there are many possibilities, H1 is used);

– **H3**: surfaces are considered according to the increasing busy percentage (if there are many possibilities, H1 then H2 are used).

The metaheuristics using this kind of representation are combined with a simulation model for the evaluation of the criterion. In this case, the model is combined with one

of the heuristics for the assignment in order to determine the surfaces assigned to each set.

15.5. Some results

The proposed methods have been tested on 3 generated instances validated by the company and on real instances. Table 15.3 gives results obtained for a mono-shop problem with a limited number of resources (Problem 2). The instances are characterized by one assembly shop with 2 elementary surfaces in the width and 6 elementary surfaces in the length, and local resources: 10 fitters, 10 operators, 3 types of pliers with 10 units of each, 20 rails assigned to the *high area* and 20 rails assigned to the *low area* of the shop. The first instance considers 45 sets of parts, the second instance considers 90 sets of parts and the third instance 180 sets of parts. For each industrial instance, 4 cases are considered:

– **case 1**: the schedule of sets and the assignment to the surfaces are fixed. This case makes it possible to evaluate the makespan of a randomly generated solution which will be used as a reference for the evaluation of the proposed methods;

– **case 2**: the schedule of sets is fixed and the objective is to find an assignment to the surfaces. The objective of this case is to compare the proposed assignment heuristics as the schedule of the sets is fixed;

– **case 3**: the assignment of the sets to the surfaces is fixed and the objective is to find a schedule of sets. The objective of this case is to compare the proposed scheduling methods;

– **case 4**: This case corresponds to the problem 2 and the objective is to find an assignment of the sets to the surfaces and a schedule.

For each case, we give the makespan and, for cases 2, 3 and 4, the improvement (percentage) compared to case 1. For case 2, the assignment of the sets to the surfaces is determined by one of the three heuristics H1, H2 or H3.

For cases 3 and 4, the used method is a stochastic descent (simulated annealing with zero temperature) with 50,000 iterations. The descent is combined with a simulation model for the computation of the criterion. For each case and each instance, 20 replications are realized, we give mean and standard deviation of the obtained makespan. For case 3, the neighboring system is an insertion neighboring system. For case 4, two neighboring systems are compared:

– a neighboring system which modifies the assignment, the orientation and the scheduling of the sets (noted $V(\sigma, S, M)$)

– a neighboring system which modifies the schedule (noted $V(\sigma)$) and the assignment of the sets is determined by the heuristic H3.

		Instance 1		Instance 2		Instance 3	
		Mean Cmax (standard deviation)	Impro vement (%)	Mean Cmax (standard deviation)	Impro vement (%)	Mean Cmax (standard deviation)	Impro vement (%)
Case 1		8590		16780		32990	
Case 2	H1	5420	36.9	10960	34.7	21640	34.4
	H2	5420	36.9	10960	34.7	21640	34.4
	H3	5400	37.1	10900	35.0	20980	36.4
Case 3	$V(\sigma)$	8025 (7.1)	06.6	10913 (30.5)	04.1	32091 (2.7)	02.7
Case 4	$V(\sigma, S, M)$	5366 (88.5)	37.5	10916 (125.9)	34.9	22806 (328.8)	30.9
	$V(\sigma), H3$	5100 (39.4)	40.6	10279 (57.35)	38.8	20927 (308.9)	36.6

Table 15.3. *Results for the mono-shop problem with a limited number of resources*

The simultaneous resolution of the scheduling and the assignment problem provides a better improvement of the makespan. We note that it is more interesting to fix the schedule then to determine the assignment than the inverse. If we compare the two neighboring systems, better results are obtained by the second one. The obtained results have been validated by the company.

15.6. Conclusion

The objective of this chapter was to present an industrial problem which can be linked to a MM-RCPSP and the methods proposed to solve it. From an academic point of view, the study of this problem led us to propose two extensions: RCPSP with variable demand profile and RCPSP with resource individualization. Mathematical models for these extensions have been presented for the Single Mode. Similar works on Multi Mode are in progress.

In the future, we plan to use the proposed extensions in other domains, such as in the health care systems for the scheduling of surgical operations in operating theaters.

Chapter 16

Employee Scheduling in an IT Company

This chapter describes a project scheduling problem of a French Information Technology company called *Eskape*. In this company, a project typically corresponds to a software development. Resources required for the execution of the project activities are the employees. We consider that the requirements of the activities are time-dependent, that employees have different skills and that the legal constraints dictated by the French workforce legislation have to be respected. The problem is studied using a global approach, dealing first with the predictive phase, where the project activities are scheduled and the employees planning are constructed. Under some hypotheses concerning unexpected events, a proactive approach and finally a reactive approach are proposed.

16.1. Introduction

We consider projects where activities only need human resources to be performed. In Eskape these resources correspond to developers and a project corresponds to a software development. Activities correspond to the analysis of the client requirements, development of the application, tests, etc. Each activity needs one particular skill for being performed, depending on the project requirements: analysis method, language development (C++, Java, etc.), etc. Moreover, since resources are human, specific constraints dictated by the workforce legislation have to be taken into account.

Chapter written by Laure-Emmanuelle DREZET, Jean-Charles BILLAUT.

The problem consists of finding a feasible solution, it means an assignment of the project activities to the employees, without violating legal constraints or project constraints and optimizing a given criterion.

However, during the project life, some unexpected events may occur and modify the forecasted schedule: one employee can be absent, sick or may have been called urgently by a customer; activities can take more time than it was forecasted, requirements of activities can vary, the project definition may have changed since the beginning of the project, etc. We propose three approaches to solve this problem: a predictive approach that does not consider such perturbations, a proactive approach that considers a knowledge on some unexpected events and that builds a robust planning and finally a reactive method for updating schedules of the employees when unexpected events occur.

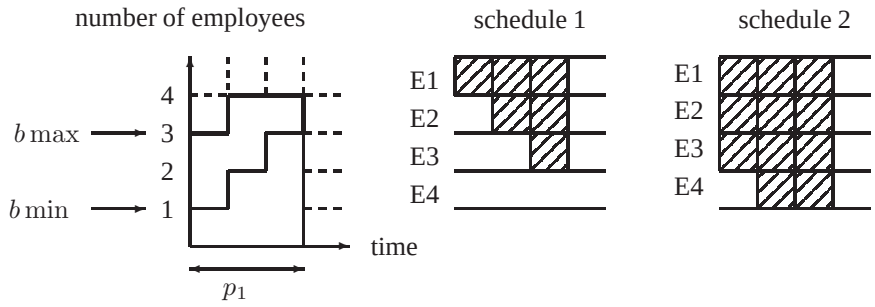
16.2. Problem presentation and context

At the beginning, the project is decomposed by the project manager into activities and each activity duration is estimated. Every two weeks or every month, all the project managers of the company give the minimal and maximal resource requirements for the project activities that have to be scheduled in the considered time horizon. Employees timetables are built every month or every two weeks for respecting these requirements. If unexpected events occur, the employees' timetables have to be updated. In this reactive phase, the considered time horizon is shorter than in the predictive or the proactive phase.

We suppose that n activities have to be scheduled on m resources that are employees. To each activity A_i ($1 \leq i \leq n$) is associated a release time r_i, a due date d_i and a duration p_i (number of periods). In contrast with the traditional RCPSP, the activity requirements are supposed to be time-dependent [POD 02]. More precisely, the requirements of activity A_i are given by two histograms ($b\min_{i,t}$ and $b\max_{i,t}$) corresponding to the minimum and maximum amount of resource needed to perform activity A_i during each time period t, $1 \leq t \leq p_i$. For example, activity A_1 of Figure 16.1 needs between 1 and 3 employees during its first processing period, between 2 and 4 employees during its second processing period and between 3 and 4 employees during its last processing period. A part of the problem is to determine the amount of resources to assign to each activity each time period.

We consider that an activity can be preempted, if it corresponds to a change in the resource assignment (see schedule 3, Figure 16.2). However, it is not possible to preempt an activity if it increases its duration (see schedule 4, Figure 16.2).

Let assume that activity A_i is assigned to employee E_j during time period t. If this activity is not assigned to this employee during time period $t+1$ and if the requirements of A_i do not strictly decrease at period $t+1$, we say that the assignment has



Activity A_1 minimal and maximal requirements

Figure 16.1. *Example of the resource requirement of one activity*

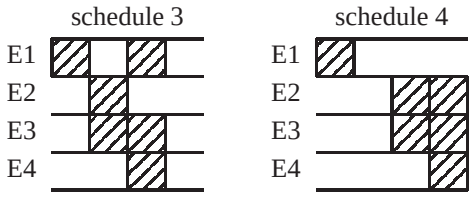


Figure 16.2. *Examples of employees' assignment to activity A1*

moved. For instance, schedule 3 in Figure 16.2 contains two assignment moves. The total number of assignment moves is bounded.

Some particular workload constraints dictated by the maintenance contracts have also to be taken into account. Indeed, a maintenance contract signed with a client can force the organization to have, on specified time periods, one or more employees able to solve specific problems. This type of workload is called *skill requirements.* The number of employees skilled to perform activity A_i that need to be present during time period t ($1 \leq t \leq T$) has to be comprised between a minimal number $b' \min_{i,t}$ and a maximal number $b' \max_{i,t}$. These employees have to be present, but can be assigned to other activities than A_i. These requirements are specified on the whole planning horizon and do not depend on the execution periods of A_i. Figure 16.3 presents an example that respects skill requirements.

Since we consider that the project requires human resources, we have to respect specific constraints for each employee such as: minimal and maximal numbers of working periods per day; maximal daily amplitude of a working day (difference between the finishing time and the starting time, including breaks); and maximal

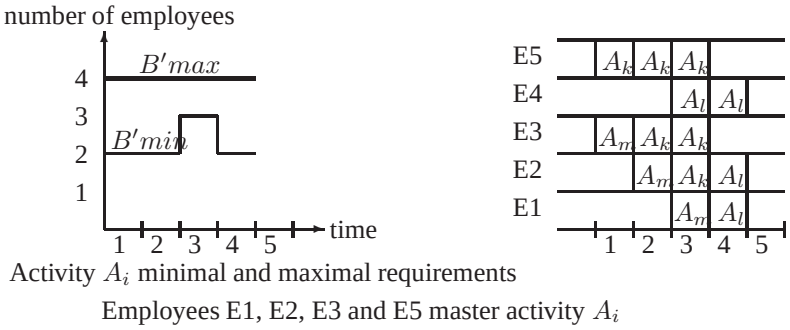


Figure 16.3. *Minimal and maximal skill requirements($B'min$ and $B'max$) of the activity A_i and assignment of employees to three activities A_k, A_l and A_m*

number of activities changes during the working day. The constraints on the amplitude and on the duration are called *legal constraints*, they are imposed by the working legislation.

The planning horizon is equal to several weeks for the predictive and the proactive phase and to one day in the reactive phase. The planning horizon is decomposed into time periods of equal length. These time periods are also determined so that no modification in the skill requirements or of the employees assignment is needed during one time period.

Each employee masters one or several skills, i.e. can perform one or several activities, but each activity of the project requires exactly one skill. The objective, in the three phases, is to find a schedule of the project activities and to determine which resources are the most appropriate to process each activity on each period.

Our problem is both a project scheduling problem and a workforce scheduling problem. For years, the research on workforce scheduling problems has been quite far from the research on project scheduling problems. The first approaches on workforce scheduling problems only consider the presence/absence of workers (see for instance the employee timetabling problem [MEI 03], [MEI 97]). Few papers study the integration between project scheduling and employee timetabling, but with a great number of simplifications ([BAI 95], [ALF 97]). The Course Scheduling Problem (CoSP) can be seen as a special RCPSP. The objective of this problem is often to find a feasible schedule. In [HAA 98] the authors use a CoSP formulation to model the practical problem of scheduling training courses in an airline company. The objective is to find a feasible planning for the courses and the trainers that maximizes the benefits. Several complex types of constraints are considered: precedence constraints, time constraints (a task can only be scheduled during the night, for example) and constraints on resources (skills of the trainers, etc.). The planning horizon is one year. Note that the CoSP

is often oversimplified to be solved. These methods cannot be used for solving the problem we consider here.

Several approaches can be used [DAV 00].

The first – *predictive* – approach consists of finding a solution to the problem, without considering that unexpected events could occur. The objective of the predictive algorithm is to find a *feasible* solution, it means without constraint violations, that minimizes the maximum lateness of the project, denoted by $L_{max} = \max_{i \in N}(cm_i - d_i)$, with N the set of activities and cm_i the completion time of activity A_i.

The second – *proactive* – approach consists of finding a solution to the problem, considering that some unexpected events could occur. The objective of the proactive algorithm is to find a feasible solution that maximizes the solution robustness. Many kinds of unexpected events may occur: the lateness or the duration increase of some activities, the unavailability of one or several resources (absenteeism, sick leave, etc.). A robustness measure is proposed. For more details on predictive approaches dedicated to project scheduling problems see [LEU 03].

The third – *reactive* – approach consists of finding a solution to the problem, considering that the project has already started and that unexpected events occur. The objective of the reactive algorithm is to find a feasible solution by updating employees' timetables that minimizes the number of employees having a modified timetable. This algorithm has to run fast and the considered time horizon is equal to several days.

16.3. Real life example

Projects that need to be scheduled in Eskape correspond mainly to software product development. These projects can be decomposed into four different phases: (1) analysis, (2) development, (3) tests and (4) install.

The following example corresponds to the development of a software application that allows a company to manage the planning of its employees. This tool is used to record the progress of the employees within the company's departments and to help managing their skills. This application has been integrated into an existing intranet application.

The activities of the project are listed in Table 16.1 and their precedence relationships are given in Figure 16.4. The project graph contains 30 nodes: 27 represent the activities of the project, three nodes (S, B and P) correspond to dummy activities.

At any time, the client may ask for information on the project completion or on the application functionalities. It is therefore important for at least one employee that

Activity	Description
	Phase 1: analysis
A_0	Existing situation analysis
A_1	Book of specifications study
A_2	Functional decomposition
A_3	UML object model development
A_4	Database creation
	Phase 2: development (Windows OS)
A_5	Creation/update/delete of information concerning employees
A_6	Consultation of employees information
A_7	Creation/update/delete of skills
A_8	Consultation of skills information
A_9	Link between skills and employees
A_{10}	Consultation of company departments information
A_{11}	Consultation of company departments information
A_{12}	Link between employees and company departments
A_{13}	Link between skills and company departments
	Phase 3: tests
A_{14}	Tests on employees treatments
A_{15}	Tests on skills treatments
A_{16}	Tests on company departments treatments
A_{17}	Tests on skills/employees relations treatments
A_{18}	Tests on company departments/employees relations treatments
A_{19}	Tests on skills/company departments relations treatments
	Phase 4: install
A_{20}	Database update
A_{21}	Source files cripting
A_{22}	Source files install on server
A_{23}	Application settings
A_{24}	Application data initialisation
A_{25}	User rights creation
A_{26}	Training

Table 16.1. *Project activities*

masters activity A_{26} or A_1 to be present (employees mastering these activities have a complete knowledge of the project and its realization).

Table 16.2 summarizes the activities characteristics. The last column shows the minimal and the maximal activities requirements. These requirements are represented by two histograms and stored in two vectors.

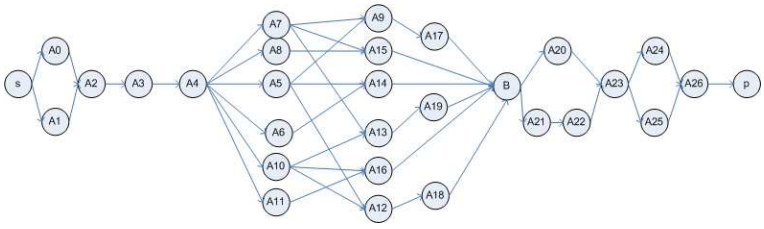


Figure 16.4. *Precedence graph*

A_i	p_i	r_i	d_i	Minimal requirements	Maximal requirements	A_i	p_i	r_i	d_i	Minimal requirements	Maximal requirements
A_0	4	0	5	(1,1,1,1)	(2,2,2,2)	A_{13}	2	10	22	(1,1)	(2,2)
A_1	4	0	5	(2,2,1,1)	(2,2,2,2)	A_{14}	3	15	23	(1,2,3)	(3,3,3)
A_2	2	0	7	(1,2)	(2,2)	A_{15}	3	15	23	(2,2,3)	(5,5,5)
A_3	4	5	10	(2,1,1,2)	(2,1,1,2)	A_{16}	2	15	23	(1,2)	(5,5)
A_4	2	0	15	(1,1)	(1,1)	A_{17}	3	17	25	(1,2,2)	(5,5,5)
A_5	4	10	20	(1,1,1,1)	(3,3,3,3)	A_{18}	3	17	25	(1,2,2)	(5,5,5)
A_6	2	10	20	(1,1)	(3,3)	A_{19}	3	17	25	(1,1,1)	(5,5,5)
A_7	4	10	20	(1,1,1,1)	(3,3,3,3)	A_{20}	2	20	30	(1,1)	(1,1)
A_8	2	10	20	(1,1)	(3,3)	A_{21}	1	20	30	(1)	(1)
A_9	2	10	22	(1,1)	(2,2)	A_{22}	1	20	30	(1)	(1)
A_{10}	4	10	20	(2,1,1,2)	(3,3,3,3)	A_{23}	1	20	30	(1)	(1)
A_{11}	2	10	20	(1,2)	(3,3)	A_{24}	1	20	30	(1)	(1)
A_{12}	2	10	22	(1,1)	(2,2)	A_{25}	1	20	30	(1)	(2)
A_{13}	2	10	22	(1,1)	(2,2)	A_{26}	4	20	35	(2,2,1,1)	(2,2,2,2)

Table 16.2. *Characteristics of the project activities*

Five employees, listed in Table 16.3, can be assigned to the project. For each employee, this table indicates the legal constraints to respect: the minimal and maximal ($Nmin$ and $Nmax$) number of worked periods per day, the maximal daily amplitude (Amp) and the maximal number of activities changes ($Nchts$). Table 16.3 also shows the skills mastered by employees. A working day is split into six two-hours periods.

The first problem consists of finding for each activity a starting time and an assignment to employees, such that the project maximum lateness is minimized, respecting constraints on activities (precedence, requirements, earliest starting and latest finishing times), resource constraints (skills, legal constraints, maximal number of activities changes) and skills requirements. Figure 16.5 represents a solution to the problem.

Resource	Skills	$Nmin$	$Nmax$	Amp	$Nchts$
HC	A0 -> A4, A14 -> A19	2	5	6	3
LED	A0 -> A26	1	4	6	5
GR	A0 -> A25	2	6	6	5
GB	A0 -> A19	1	6	6	3
AN	A5 -> A13, A21 -> A23	1	4	5	5

Table 16.3. *Resources available for the project*

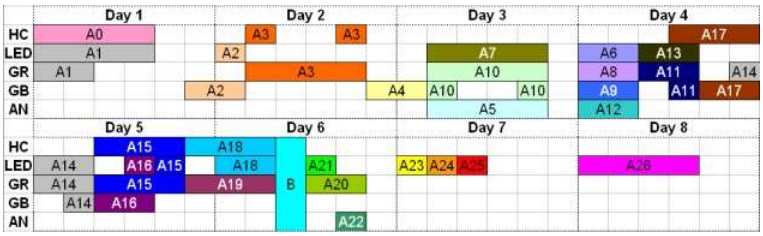


Figure 16.5. *A feasible schedule of the project*

This solution makes it possible to obtain a schedule for each employee and for each activity of the project. However, whatever the quality of the schedule and the assignment, remember that this solution will probably be modified due to unexpected events.

16.4. Predictive phase

16.4.1. *Greedy algorithms*

We propose a basic greedy algorithm based on the following scheme. Let U be a list of candidate activities, i.e. activities having their predecessors already scheduled. This list is sorted according to a priority rule denoted by W_U. For each activity A_i an earliest starting time est_i and a latest completion time lct_i are computed. We denote by Emp_i the set of employees available in the interval $[est_i, lct_i]$ and mastering the skill required for performing activity A_i. The employees of Emp_i are sorted according to a priority rule denoted by W_E.

An assignment to A_i is a subset of Emp_i. Enumerating all the subsets of Emp_i is not possible. Thus, a list of n_i assignments is built (n_i arbitrarily fixed), assuming that the assignments that do not respect the work legislation are put at the end of the list. Each assignment is evaluated by the lateness of activity A_i and the one with the minimum L_i value is selected. Set U is updated and the process iterates until U

is empty. Obviously, the solutions returned by this algorithm are neither necessarily optimal nor feasible since some constraints are not taken into account.

Different priority rules W_U have been tested to sort activities of set U: Earliest Due Date (EDD), sum of minimal resource requirements, minimal number of successors, minimal slack, Max-Tail, Max-Tail$_i$ being the length of the longest path between activity A_i and the sink node of the project graph.

Similarly, different priority rules W_E have been tested to sort the employees that can perform A_i: the number of employees' skills in a non-decreasing order; by first choosing the employees for which the assignment fits the best to the skill-requirements for the considered time window; by first choosing the employees available that can be assigned to the considered activity; by first choosing the employees available that can be assigned to the considered activity, taking the holes in timetables into account (EarliestHole); or by using the EarliestHole rule with a right-shift of activities to find a legal solution.

When all the activities have been scheduled, a repairing phase is executed to increase the number of employees assigned to each activity of the project. The aim is to decrease the number of skill-requirements violations without increasing the number of violated legal constraints. This is done by assigning some additional employees to activities when it is both possible and required.

16.4.2. *Tabu search algorithm*

The aim of the tabu search algorithm is to enhance the solution returned by the greedy algorithm. A state in this tabu search algorithm is composed of a vector giving for each activity of the project its starting and ending time periods, and by the assignment matrix of employees to activities, for each time period.

Four types of moves are used in this method, based on activity moves and swaps or on employee assignments moves or swaps: (1) the change of the assignment of one employee on one time period; (2) the swap between the assignments of two employees on the same time period; (3) the left or right shift of the starting time of one activity for one or several time periods; and (4) the swap between two activities.

In a given state, the generated moves are those that lead to neighbors respecting precedence constraints between activities and skill constraint. As the size of the generated neighborhood may be very large, all the neighbours of the current solution are not generated at each iteration: moves based on employees are only generated from a selected time period and the number of time periods of the right and left shifts are bounded.

A solution S is evaluated using two criteria: the number of violated constraints and the maximum lateness. These two objectives are considered in the lexicographic order.

16.5. Proactive phase

In a proactive approach, the schedule is built considering some knowledge on the unexpected events that could occur and modify the current schedule. In our problem, we consider several types of unexpected events. Moreover, the company is not necessarily interested to have a solution optimizing the maximum lateness, if this solution has to be totally recomputed when the first unexpected event occurs. Thus we search for solutions minimizing the total number of violated constraints and among all these solutions, we search for the most robust solution. For more details on robust scheduling see [BIL 05].

The unexpected events that may occur concern both employees and activities. Two main types of perturbations can be distinguished: the absenteeism of employees and the modification of the duration of activities. We consider that each employee may be absent. This unavailability can correspond to the assignment of these employees to activities belonging to other projects or to unexpected leaves. Two data can be associated with each employee: a probability of absenteeism and a maximal absence duration. When perturbations affect an activity, we consider that the duration of this activity is modified. In reality, when this kind of perturbation occurs, it corresponds to an increase of the duration of the activity. This perturbation is often due to mistakes made by the project manager when he/she estimates the duration of the activities. Thus, we consider here that the duration of the perturbated activity is greater than the normal duration. Therefore, two data are associated with each activity: a probability of perturbation and a maximal duration of perturbation.

We define two robustness measures for a schedule S: ρ_S^P which is based on employees, and ρ_S^A which is based on activities. ρ_S^P is computed using some information on each employee j, as the absence probability p_{ind_j} and the maximal duration of absence d_{ind_j}. ρ_S^A is computed using some knowledge on the possible disturbances for each activity: the probability of perturbation and the maximal duration of perturbation. The total robustness of schedule S is given by a linear combination of ρ_S^A and ρ_S^P (with $\alpha = 0.5$).

$$\rho_S^P = \min_{j \in J} -(workload_j + d_{ind_j}) \times p_{ind_j}$$
$$\rho_S^A = \min_{i \in I}(slack_i - d_{all_i}) \times p_{all_i}$$
$$\rho_S = -(1-\alpha) \times \rho_S^A - \alpha \times \rho_S^P$$

It is important to note that the robustness metric and the maximum lateness criterion are conflicting in this context. The predictive algorithm has been adapted and two new priority rules have been defined: ROBACT based on the slack of activity A_i and the probability of perturbation, and RobEmp based on the load of employee E_j and the probability of absenteeism. The selection of the best assignment uses the robustness measure instead of the lateness measure.

The predictive tabu search algorithm has also been adapted to evaluate the quality of each neighbor with its number of violated constraints and its robustness.

16.6. Reactive phase

In this phase, the company must face absenteeism of some employees or modification of the duration of some activities. It is supposed that the timetables of employees are known and that due to unforeseen events we have to modify some of these timetables. Since the algorithm has to return a solution quite quickly, the considered time horizon has been reduced to one day. On the other hand, since more information are available during this phase than during the previous one (predictive or proactive), we can be more precise and the length of each time period has been reduced in the reactive phase. As beginning and ending periods of activities are data of this problem, precedence relationships between activities are implicitly known and are not explicitly considered in this phase.

A weight is associated with each employee if his timetable has already been modified recently. The aim is to ensure ourselves that the modifications do not always concern the same employees. The objective of the problem is to update timetables, such that the new constraints are satisfied and the weighted number of modified timetables is minimized.

A tabu search algorithm has been developed for this problem, inspired by the tabu search algorithm of [MUS 04]. The initial solution of the method corresponds to the initial employees timetables. This solution can violate either the skills requirements or the activities requirements constraints. As for the predictive tabu search algorithm, criteria are minimized in a lexicographical order: first the number of violated constraints and then the number of modified timetables. The tabu list contains moves that were selected during the previous iterations. As the size of the neighborhood may be very large, only a reduced number of neighbors (limited by a parameter) are generated. The moves depend on the kind of violated constraints and different diversification methods have been tested.

16.7. Computational experiments

16.7.1. *Predictive and proactive algorithms*

To evaluate the performances of the algorithms, we have adapted the instances of the PSPLIB (see [KOL 97, PSPib] and Chapter 7). The modified instances are generated using the precedence graph of the PSPLIB to link activities and to find their release times and due dates. The activities requirements are computed using the original requirement of activities and the number of employees skilled to perform the activities. We assume that each employee masters 70% of the skills in all the instances we consider.

Depending on the priority rules , 32 heuristic algorithms have been tested in four categories (some heuristic algorithms use a second priority-rule to break ties). The results show that two heuristic algorithms outperform the others for the robustness measure. We can also notice that the robust algorithms make it possible to respect more constraints (only legal and skill-requirements constraints may be violated) than the predictive algorithms. All the problem instances have been solved by the heuristic algorithms in less than one second.

The predictive and the proactive tabu search algorithms have also been evaluated, using the results given by the heuristic algorithms. Initial solutions of the predictive tabu search algorithm are provided by the best predictive greedy heuristic. Table 16.4 presents the results of the tabu search algorithms. Column “# of viol. constr." is the average number of violated constraints, column L_{max} and ρ_S indicate the average value of these criteria.

	# of viol. constr.	L_{max}	ρ_S
Predictive approach	**124.43**	**85.33**	28.30
Proactive approach	128.10	88.55	**25.19**

Table 16.4. *Comparison of the predictive and the proactive tabu search algorithms (with respect to the number of violated constraints)*

These results show that the predictive method leads to results with a better L_{max} value than the proactive method, but a greater value of ρ. However, the difference between the robustness of “robust" solutions obtained by a proactive method and the robustness of “non-robust" solutions is not so important. This is because in the proactive and predictive methods, the real first objective to minimize is the number of violated constraints. After this first objective, comes the robustness (for the proactive method) or the maximum lateness (for the predictive method).

16.7.2. *Reactive algorithm*

To evaluate the performance of the reactive algorithm, we have generated instances based on PSPLIB library. More details on this generation are given in [DRE 05]. Three sets of instances were build : the first is composed by instances in which unexpected events on activities occurred; the second is composed of instances in which unexpected events on employees occurred; the last is composed of instances in which both types of unexpected events occurred. All the three sets are composed of 96 instances, divided into five subsets, depending on the number of employees and activities.

The results of the tabu search algorithm have been compared to the results of an integer linear programming model, solved by CPLEX solver [DRE 05]. With a maximum CPU time of 5 minutes, the tabu search algorithm always makes it possible to obtain feasible solutions, while CPLEX is not efficient for solving medium-size (only 15% to 42.9% feasible solutions) and large-size instances (between 0% and 12.5% feasible solutions).

16.8. Conclusion

In this chapter, we consider a particular RCPSP with human resources and time-dependent activity requirements. The aim is to respect legal constraints and skill-requirements and to minimize a quality measure associated to the project. This problem occurs in a French IT company, and the proposed algorithms have been quite integrated in the company's Intranet. The problem has been studied from a predictive, proactive and reactive points of view.

For the predictive and proactive approaches, different priority rules have been tested and tabu search algorithms have been developed. These methods make it possible to reduce the total number of violated constraints and lead to good-quality solutions. For the reactive phase, a tabu search algorithm and a MILP have been developed. The heuristic method leads quickly to feasible solutions, while the MILP is quite inefficient.

Our future work will consist of developing new robustness measures, considering explicitly the resource constraints when computing slacks, for example. All the heuristic algorithms can also be used to generate a part of an initial population for a genetic algorithm that may reach a better compromise between the number of violated constraints and the robustness of the solution (for the proactive phase) or the number of modified timetables (for the reactive phase).

Chapter 17

Rolling Ingots Production Scheduling

In this chapter we study a real-world scheduling problem arising in the context of a rolling ingots production in aluminium industry. In the first part of the chapter we review the production process and discuss the constraints that have to be observed when scheduling a given set of production orders on the production facilities. We then show how this problem can be modeled as a RCPSP with minimal and maximal time lags between activities and different kinds of renewable resources. A solution procedure of type branch-and-bound is sketched in the second part. The basic principle consists of relaxing the resource constraints by assuming infinite resource availability. Resulting resource conflicts are iteratively resolved by introducing precedence relationships among activities competing for the same resources.

17.1. Introduction

We consider the production of rolling ingots, i.e. ingots of a certain aluminium alloy in rectangular form. These ingots are the starting material for the rolling of sheet, strip and foil. The production process can be described as follows (see section 1.4 in [KAM 99] and Figure 17.1).

In a melting furnace, which is called a potroom, the ingredients composing the alloy are smelted in an electrolytical process. In general, several different potrooms are available at a production plant. One or several casting units belong to each potroom. A casting unit consists of a holding fixture for a mould, a retractable hydraulic cylinder named stool, and a stool-cap, which closes the bottom of the mould at the

Chapter written by Christoph SCHWINDT, Norbert TRAUTMANN.

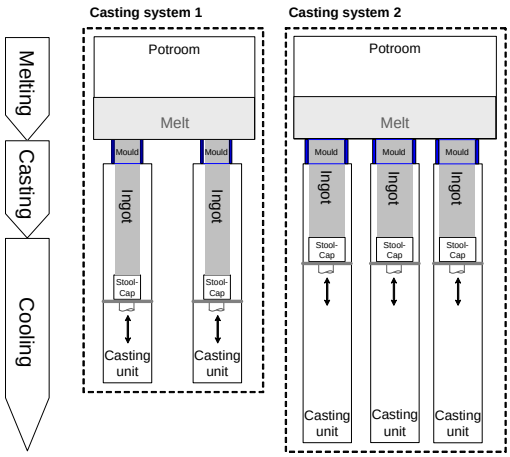


Figure 17.1. *Production plant with two casting systems*

start of the casting process. The mould determines the cross-section of the ingot. A potroom together with its casting units is called a casting system. When the ingredients have been smelted, the melt is cast through the mould. The casting has to be started immediately after the end of the melting and must be completed jointly at all casting units of a casting system. As soon as the metal in the mould begins to solidify, the stool is lowered and the resulting ingot is cooled by spraying water. The maximum stroke of the stool determines the maximum cast length, which implies that not every ingot can be produced on each casting unit. However, all casting units of one casting system are of the same height, and all ingots produced within one casting have the same length. It is not necessary to use all casting units during a casting process nor to use their full height. After the completion of the casting, the ingot stays some time in the casting unit for cooling. When passing to the casting of an ingot with a different cross-section, the mould of the casting unit has to be replaced. The changeover can only be performed when no casting is in process. Moreover, due to the limited availability of cranes, only one mould per potroom can be exchanged at a time.

The production scheduling problem can be stated in the following way. Given a set of production orders for ingots characterized by their measurements and alloys, the problem consists of computing a feasible production schedule with minimum makespan. Approaches for related problems have been proposed. Fleischmann and Jess [FLE 85] present a simulation model of a rolling ingots production, which is used for strategic planning and short-term scheduling. Kempf *et al.* [KEM 98] discuss a scheduling problem which arises in waver production. The oven and the boards used for the quality control of the integrated circuits respectively correspond to a potroom and the moulds in our problem. Harjunkoski and Grossmann [HAR 01] combine mixed-integer linear programming with a decomposition strategy to solve scheduling

problems occurring in steel plants. An extended version of the present chapter can be found in [SCH 03].

The remainder of this chapter is organized as follows. In section 17.2 we explain how the production scheduling problem can be formulated as a multi-mode project scheduling problem with minimal and maximal and different kinds of renewable resources. In section 17.3 we briefly sketch the basic principle of a solution procedure of the branch-and-bound type.

17.2. Project scheduling model

17.2.1. *Activities and temporal constraints*

Let J be the given set of production orders or jobs $\ell = 1, \ldots, \nu$ for individual ingots. Each job ℓ consists of the four operations melting the ingredients in the potroom, changeover of the mould, casting the melt from the potroom into the casting unit, and cooling the ingot in the casting unit. For what follows we consider the execution of the jobs $\ell \in J$ as a project, where we associate the operations of jobs $\ell \in J$ with the n activities $A_1 \ldots, A_n$ of the project. We define the activities in such a way that during their execution they have constant requirements for the production facilities. Accordingly, the set of activities $\mathcal{A}$ consists of three subsets $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ which for every job ℓ contain the three activities corresponding to the melting plus casting in a potroom, the exchange of the mould, and the casting plus cooling in a casting unit of the casting system (see Figure 17.2). The dummy activities A_0 and A_{n+1} represent the start and the completion of the production. The duration of activity A_i is denoted by p_i, where $p_0 = p_{n+1} = 0$. The duration p_i of a changeover activity $A_i \in \mathcal{A}_2$ depends on the sequence in which the jobs are processed on the casting unit. If job ℓ and the job ℓ' preceding ℓ on the casting unit require different moulds, p_i equals the setup time s_ℓ for installing the mould belonging to ℓ. Otherwise, the mould need not be replaced, and p_i is equal to zero.

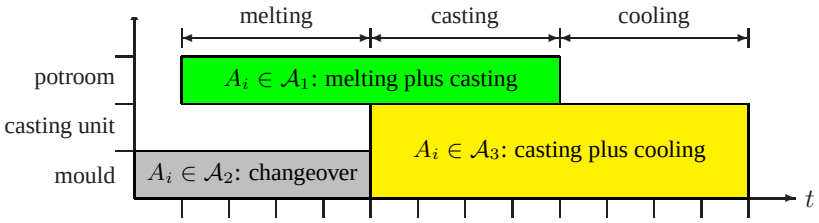


Figure 17.2. *Activities A_i of the project*

Since the jobs can be performed in alternative casting systems, we define a set M_i of alternative **execution modes** m for each activity $A_i \in \mathcal{A}$ along with corresponding

binary decision variables x_i^m, which are equal to one if activity A_i is executed in mode m and equal to zero, otherwise. Vector $x = (x_i^m)_{A_i \in \mathcal{A}, m \in M_i}$ is referred to as a mode assignment. The *mode-assignment constraints*

$$\sum_{m \in M_i} x_i^m = 1 \quad (A_i \in \mathcal{A}) \tag{17.1}$$

say that each activity has to be carried out in exactly one mode. In addition, we have to ensure that all activities belonging to one and the same job ℓ are carried out in the same casting system. Let $\mathcal{A}^\ell \subseteq \mathcal{A}$ be the set of activities belonging to job ℓ. The condition is formulated as the *mode-identity constraints*

$$\sum_{m \in M_i \cap M_j} x_i^m x_j^m = 1 \quad (\ell \in J;\ A_i, A_j \in \mathcal{A}^\ell). \tag{17.2}$$

Note that in our case it holds that $M_i = M_j$ for all $A_i, A_j \in \mathcal{A}^\ell$ so that (17.2) could also be written in the form $(x_i^m)_{m \in M_i} = (x_j^m)_{m \in M_j}$.

The following **temporal constraints** arise from technological requirements. Consider the melting plus casting activity A_h, the changeover activity A_i, and the casting plus cooling activity A_j of a job $\ell \in J$. By definition, activities A_h and A_j are pulled tight. Let c_ℓ denote the cooling time for job ℓ. Then we have a *minimal time lag* $\delta_{hj}^{min} = p_h - p_j + c_\ell$ and an equally large *maximal time lag* $\delta_{hj}^{max} = p_h - p_j + c_\ell$ between the starts of activities A_h and A_j. In addition, the mould has to be installed before the casting. This precedence relationship between activities A_i and A_j gives rise to the minimal time lag $\delta_{ij}^{min} = p_i$. The conditions that no activity is started before the production start nor completed after the production end can be ensured by introducing the minimal time lags $\delta_{0h}^{min} = \delta_{0i}^{min} = 0$ and $\delta_{j(n+1)}^{min} = p_j$.

The activity-on-node network G depicted in Figure 17.3 illustrates the activities and prescribed time lags of the project. Each activity A_i is identified with a node of the network. Thus, the node set V of the network coincides with the set $\mathcal{A} \cup \{A_0, A_{n+1}\}$ of all real and dummy activities. For each minimal time lag δ_{ij}^{min}, network G contains an arc (A_i, A_j) with initial node A_i, terminal node A_j, and weight $\delta_{ij} = \delta_{ij}^{min}$. A maximal time lag δ_{ij}^{max} is represented as an arc (A_j, A_i) with initial node A_j, terminal node A_i, and weight $\delta_{ji} = -\delta_{ij}^{max}$.

Now let $S_i \geq 0$ denote the decision variable providing the start time of activity $A_i \in V$. A vector (S, x) with $S = (S_i)_{A_i \in V}$ is called a (production) *schedule*. By convention, we set $S_0 := 0$, and thus S_{n+1} equals the production makespan. We say that a schedule (S, x) is *mode-feasible* if mode assignment x satisfies the mode-assignment and mode-identity constraints (17.1) and (17.2). Schedule (S, x) is called *time-feasible* if S complies with the *temporal constraints*

$$S_j - S_i \geq \delta_{ij} \quad ((A_i, A_j) \in E) \tag{17.3}$$

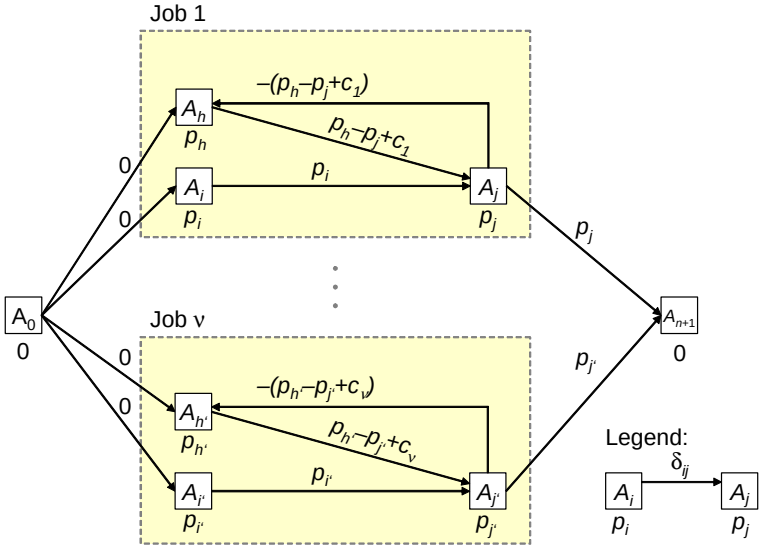


Figure 17.3. *Activity-on-node network G*

where E is the arc set of network G. For $\delta_{ij} = p_i$, inequality (17.3) is also called a *precedence constraint*. It is well-known that a time-feasible schedule exists if and only if G does not contain any cycle of positive length (see, e.g., [BAR 88]). In our case, the latter condition is always met because all cycles have length zero.

17.2.2. *Resource constraints*

In this section we are concerned with the resource constraints of the production scheduling problem. Basically, all production facilities (potrooms, casting units, and moulds) correspond to **renewable resources**. The set of renewable resources is denoted by $\mathcal{R}$, and B_k stands for the capacity of renewable resource $R_k \in \mathcal{R}$.

An activity $A_i \in \mathcal{A}$ takes up b_{ik}^m units of resource $R_k \in \mathcal{R}$ if A_i is executed in mode $m \in M_i$. By $\mathcal{A}_k(x) := \{A_i \in \mathcal{A} \mid \sum_{m \in M_i} b_{ik}^m x_i^m > 0\}$ we denote the set of all activities being executed on resource R_k given mode assignment x. We call a schedule (S, x) *capacity-feasible* if for each renewable resource $R_k \in \mathcal{R}$, no more than B_k units of R_k are required simultaneously, i.e.,

$$\sum_{\substack{A_i \in \mathcal{A}_k(x): \\ S_i \le t < S_i + p_i}} \sum_{m \in M_i} b_{ik}^m x_i^m \le B_k \quad (R_k \in \mathcal{R};\ t \ge 0) \tag{17.4}$$

In a potroom, only one alloy can be melted at a time. This condition can be taken into account by using the concept of **batching machines** and incompatible job families discussed in literature on machine scheduling (see, e.g., [UZS 95] or [POT 00]). Let $\mathcal{R}^\beta \subseteq \mathcal{R}$ denote the set of batching machines and let φ_i designate the family of activity A_i. Family φ_i identifies the alloy and length of the ingot belonging to activity A_i. Recall that in a casting system, two melting plus casting activities $A_i, A_j \in \mathcal{A}_1$ can only be processed in parallel if the corresponding ingots are of the same alloy and length since the melt is cast in all casting units simultaneously. A batching machine has to be operated in such a way that the activities running in parallel must belong to the same family and must be started at the same time. Accordingly, a schedule (S, x) is called *batching-feasible* if

$$S_i = S_j \text{ and } \varphi_i = \varphi_j \qquad (A_i, A_j \text{ with } [S_i, S_i + p_i[\cap [S_j, S_j + p_j[\neq \emptyset \text{ and } A_i, A_j \in \mathcal{A}_k(x) \text{ for some } R_k \in \mathcal{R}^\beta) \quad (17.5)$$

Each potroom is modeled as a batching machine $R_k \in \mathcal{R}^\beta$ whose capacity B_k equals the number of casting units belonging to the casting system. For the resource requirements we have $b_{ik}^m = 1$ if $A_i \in \mathcal{A}_1$ and if in mode m, activity A_i is executed in the potroom of resource R_k.

Next, we consider the changeover of the mould and the casting process. As previously stated in section 17.1, the casting requires the mould of the desired cross-section to be installed on the casting unit. Obviously, it is not possible to use the installed mould for processing another ingot before the casting has been completed. To model the latter requirement, we introduce a set $\mathcal{R}^\alpha \subseteq \mathcal{R}$ of **allocatable resources**. The b_{ik}^m units of allocatable resource $R_k \in \mathcal{R}^\alpha$ processing an activity $A_i \in \mathcal{A}$ in mode $m \in M_i$ remain occupied from the start of a given *allocating activity* $A_{h(i)}$ up to the completion of activity A_i, where for simplicity we assume that $A_{h(i)} \notin \mathcal{A}_k(x)$.

A schedule (S, x) is called *allocation-feasible* if at any point in time, no more than B_k units of a resource $R_k \in \mathcal{R}^\alpha$ are allocated, i.e.,

$$\sum_{\substack{A_i \in \mathcal{A}_k(x): \\ S_{h(i)} \leq t < S_i + p_i}} \sum_{m \in M_i} b_{ik}^m x_i^m \leq B_k \quad (R_k \in \mathcal{R}^\alpha;\ t \geq 0). \qquad (17.6)$$

Moulds of the same type form an allocatable resource $R_k \in \mathcal{R}^\alpha$ with capacity B_k being equal to the number of moulds of that type. Each casting plus cooling activity A_i requires one unit of the allocatable resource $R_k \in \mathcal{R}^\alpha$ corresponding to the mould type needed, i.e., $b_{ik}^m = 1$ for all $A_i \in \mathcal{A}_3$ and all $m \in M_i$. The mould is allocated to activity A_i at the start of the corresponding changeover activity $A_{h(i)} \in \mathcal{A}_2$.

We now turn to resources that have to be changed over between consecutive activities. Let $\mathcal{R}^\gamma \subseteq \mathcal{R}$ denote the set of those **changeover resources**. Given a schedule (S, x), let $A_i, A_j \in \mathcal{A}_k(x)$ be two activities sharing some changeover resource $R_k \in \mathcal{R}^\gamma$ and let ϑ_{ij}^k denote the time needed for changing over a unit of resource R_k from A_i to A_j. ϑ_{0i}^k and $\vartheta_{i(n+1)}^k$ are the initial setup and terminal tear-down times for resource R_k. We assume that the changeover times satisfy the *weak triangle inequalities*

$$\vartheta_{hi}^k + p_i + \vartheta_{ij}^k \geq \vartheta_{hj}^k \quad (R_k \in \mathcal{R}^\gamma;\ A_h, A_i, A_j \in \mathcal{A}_k(x) \cup \{A_0, A_{n+1}\}).$$

Now consider the set

$$P_k(S,x) = \{(A_i, A_j) \mid A_i \in \mathcal{A}_k(x) \cup \{A_0\};\ A_j \in \mathcal{A}_k(x) \cup \{A_{n+1}\};\\ S_j \geq S_i + p_i + \vartheta_{ij}^k\}$$

of all pairs (A_i, A_j) for which the time span between the completion of activity A_i and the start of activity A_j is sufficiently long for a changeover on R_k. Let f_{ij}^k for $(A_i, A_j) \in P_k(S,x)$ denote the decision variable which is equal to one precisely if A_i and A_j are to be processed consecutively on some unit of resource R_k, and equal to zero, otherwise. $f_{ij}^k = 1$ causes a changeover time of ϑ_{ij}^k time units between the completion of A_i and the start of A_j. There exists a feasible assignment of the activities $A_i \in \mathcal{A}_k(x)$ to the individual units of resource R_k observing the changeover times if and only if first,

$$\sum_{(A_j,A_i)\in P_k(S,x)} f_{ji}^k = \sum_{(A_i,A_j)\in P_k(S,x)} f_{ij}^k = 1 \quad (R_k \in \mathcal{R}^\gamma;\ A_i \in \mathcal{A}_k(x)) \qquad (17.7)$$

and second,

$$\varphi_k(S,x) := \sum_{(A_0,A_j)\in P_k(S,x)} f_{0j}^k \leq B_k \quad (R_k \in \mathcal{R}^\gamma). \qquad (17.8)$$

In section 2.13 in [NEU 03], it is shown that the problem of minimizing $\varphi_k(S,x)$ subject to equations (17.7) represents a minimum-flow problem in the precedence graph of strict order $P_k(S,x)$.

We call a schedule (S,x) *changeover-feasible* if for all $R_k \in \mathcal{R}^\gamma$ there exist flows $f^k = (f_{ij}^k)_{(i,j)\in P_k(S,x)}$ satisfying conditions (17.7) and (17.8).

We associate each group of casting units belonging to one casting system with a changeover resource $R_k \in \mathcal{R}^\gamma$ whose capacity B_k equals the number of casting units in the system. If casting plus cooling activity A_i is executed in the mode $m \in M_i$ belonging to the casting system of resource R_k, A_i requires one unit of resource R_k,

i.e., $b_{ik}^m = 1$. If jobs ℓ and ℓ' use the different mould types, the changeover time between the respective casting plus cooling activities $A_i, A_j \in \mathcal{A}_3$ on R_k is equal to the time needed for installing the mould for job ℓ', i.e., $\vartheta_{ij}^k = s_{\ell'}$. Otherwise, the changeover time is equal to zero. The duration of the corresponding changeover operation $A_h \in \mathcal{A}_2$ of job ℓ' is $p_h = \vartheta_{ij}^k$.

The last resource constraint to be modeled emanates from the technological requirements that first, only one of the casting units belonging to a casting system can be changed over at a time and second, a mould cannot be installed into any casting unit during casting or cooling. Those conditions are modeled by defining an extra renewable resource $R_k \in \mathcal{R}$ for each casting system, where again capacity B_k is chosen to be equal to the number of casting units of the casting system. Resource R_k is taken up by the changeover and casting plus cooling activities A_i and A_j of those jobs ℓ which in mode assignment x are performed on the casting system associated with R_k. We set $b_{ik}^m = B_k$ and $b_{jk}^m = 1$, where m is the mode corresponding to the respective casting system. Then constraints (17.4) ensure that no two casting units can be changed over in parallel and that up to B_k casting plus cooling operations A_j can be processed simultaneously, but no operation can be processed while one of the units is changed over.

17.2.3. *Production scheduling problem*

A schedule (S, x) that is capacity-, batching-, allocation-, and changeover-feasible is termed *resource-feasible*. A mode-, time-, and resource-feasible schedule (S, x) is called *feasible*. A schedule (S, x) is called *optimal* if (S, x) is feasible and there is no feasible schedule (S', x') with a shorter makespan $S'_{n+1} < S_{n+1}$.

To sum up, the production scheduling problem (P) for rolling ingots reads

$$\left.\begin{array}{lll} \text{Minimize} & S_{n+1} & \\ \text{subject to} & \text{(17.1) to (17.8)} & \\ & S_i \geq 0 & (A_i \in \mathcal{A}) \\ & x_i^m \in \{0,1\} & (A_i \in \mathcal{A};\ m \in M_i) \\ & f_{ij}^k \geq 0 & (R_k \in \mathcal{R}^\gamma;\ (A_i, A_j) \in P_k(S,x)) \end{array}\right\} \quad \text{(P)}$$

17.3. Solution method

The branch-and-bound procedure is based on a schedule-generation scheme presented in section 2.16 in [NEU 03]. The constraints that make the problem intractable are

1) the necessity to select a mode for each activity;

2) the limited capacity of the renewable resources;

3) the requirement that activities running in parallel on batching machines must be of the same batching type and must be started at the same time; and

4) the need for changeover activities with sequence-dependent durations.

If we relax all those constraints, the remaining problem consists of scheduling all activities subject to the temporal constraints. This temporal scheduling problem can be solved efficiently by computing the earliest start times ES_i for all activities $A_i \in V$, which coincide with the longest path lengths from node A_0 to node A_i in network G.

We start the generation scheme with the schedule (S, x) belonging to the vector $S = ES$ of earliest start times and the empty mode assignment $x = 0$. In each iteration of the algorithm the schedule (S, x) under consideration may be infeasible for the production scheduling problem due to two reasons: (i) there may be activities for which no mode has been selected so far, or (ii) one of the resource constraints may not be met. In case (i), we extend mode assignment x by selecting some mode for an activity whose mode has not been chosen yet. Exploiting the mode-identity constraints, the modes of the other activities belonging to the same job can be fixed at the same time. In case (ii), the violation of some resource constraint at a given time is resolved by introducing appropriate time lags between activities competing for the respective resource (see [SCH 03] for details). By assigning modes to activities and adding time lags, we obtain a new scheduling problem with a refined relaxation, which belongs to a reduced set of feasible modes and to an expanded network with new arcs. The selection of modes and the addition of arcs is continued until either a mode has been selected for each activity and the solution to the temporal scheduling problem provides a feasible schedule or the temporal scheduling problem is unsolvable. Figure 17.4 summarizes this schedule-generation scheme.

In the branch-and-bound algorithm, the enumeration is performed as follows. In the case of a mode selection for some activity A_i we generate one enumeration node for each mode $m \in M_i$. When dealing with a violation of a resource constraint, we enumerate the alternatives of choosing two activities A_i, A_j among the activities causing the conflict and define an appropriate minimal time lag δ_{ij}^{min} between A_i and A_j. Each time we have reached a feasible schedule or the temporal scheduling problem has become unsolvable, we perform backtracking and proceed with the investigation of unexplored enumeration nodes.

Due to the sequence-dependent need for changeovers, the resource demands over time arising from the changeover activities are not known until the activity sequences on the changeover resources have been established. That is why we follow a heuristic two-phase approach where in the first phase, while observing the changeover times, we assume that changeover activities do not require resources for their execution.

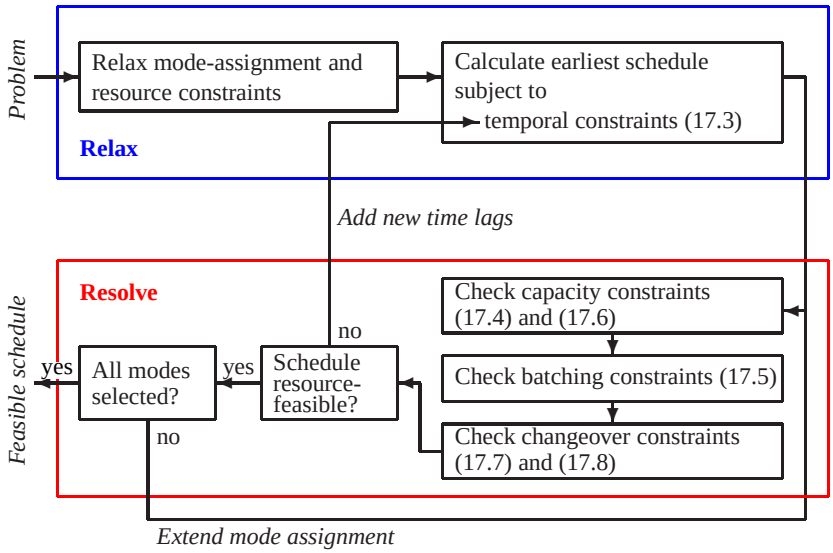


Figure 17.4. *Schedule-generation scheme of the branch-and-bound algorithm*

In the terms of our production scheduling problem the latter assumption means that we allow for parallel changeovers on casting units belonging to one and the same casting system. When we have obtained a feasible schedule for this subproblem, we fix the activity sequences on the changeover resources (and thus the durations of the changeover activities) by introducing a precedence constraint between any two consecutive activities. We then proceed with the second phase, where we apply the branching process subject to the time lags introduced.

We have evaluated the performance of a filtered beam search version of the branch-and-bound algorithm. The tests have been performed based on 15 instances including up to 50 jobs. The results for the larger instances indicate that the algorithm scales well. In particular, the makespan-per-job ratio seems to be independent of the problem size, though the CPU time limit has not been varied. Details of the performance analysis can be found in [SCH 03].

17.4. Conclusions

In this chapter we have considered the problem of scheduling the casting of rolling ingots on a production plant consisting of several casting systems. The problem is characterized by specific technological constraints such as cooling times, incompatibilities between production orders, and changeovers whose necessity depends on the sequence in which the production orders are processed on the casting units. We have

formulated the problem as a resource-constrained project scheduling model with minimal and maximal time lags between the activities and different types of resources. For solving the problem, we have proposed a branch-and-bound algorithm where we relax the resource constraints and enumerate the different alternatives of assigning execution modes to the activities and resolving resource conflicts between activities competing for the same resources. An experimental performance analysis shows that a truncated version of the algorithm is able to find good schedules within a reasonable amount of computation time.

Chapter 18

Resource-Constrained Modulo Scheduling

18.1. Introduction

Modulo scheduling is an effective instruction scheduling framework for optimizing innermost program loops, especially when compiling for Very Long Instruction Word (VLIW) processors. VLIW processors are instruction-level parallel processors [RAU 93] where the compiler is the only responsible for instruction scheduling. By comparison, superscalar processors like the Intel Pentium series dynamically reschedule instructions by hardware, based on the linear order of instructions produced by the compiler. The Lx [FAR 00] is an example of modern VLIW architecture jointly developed by Hewlett-Packard Laboratories and STMicroelectronics. This architecture is implemented by the STMicroelectronics ST200 VLIW processor family.

Modern VLIW processors are mainly used for media processing in embedded devices such as mobile, automotive, and consumer electronics. For such applications, the time taken by compiler optimizations is not an issue. Rather, the quality of the instruction schedules produced has a direct impact on the overall system cost and energy consumption, as high-quality instruction schedules enable to reduce the operating frequency given the real-time processing requirements. This allows for the use of optimal instruction scheduling techniques for VLIW processors, in particular on inner loop code where most of the processing time is spent.

Instruction scheduling for inner loops is also known as software pipelining [ALL 95]. Among the different software pipelining frameworks, *modulo scheduling* [RAU 96] is the most successful in production compilers. The modulo scheduling

---

Chapter written by Benoît DUPONT DE DINECHIN, Christian ARTIGUES, Sadia AZEM.

framework is distinguished by its focus on 1-periodic cyclic schedules with integral period, which leads to simplifications compared to the classic formulation of cyclic scheduling [HAN 95].

In this chapter, we define the *resource-constrained modulo scheduling problem* (RCMSP) as a modulo scheduling problem where the resources constraints are adapted from the renewable RCPSP. We then introduce a new time-indexed integer linear programming formulation of the RCMSP, along with the one proposed earlier by Eichenberger *et al.* [EIC 95, EIC 97]. Finally, we propose a large neighborhood search heuristic to solve difficult instances of these formulations.

18.2. The Resource-constrained modulo scheduling problem

18.2.1. *The resource-constrained cyclic scheduling problem*

A *basic cyclic scheduling* [HAN 95] is defined by a set of generic operations $\mathcal{A} = \{A_i\}_{1 \leq i \leq n}$ to be executed repeatedly, thus defining a set of operation instances

$$\{A_i^l\}_{1 \leq i \leq n}^{l \in \mathbb{N}}.$$

We call *iteration* l the set of operation instances $\{A_i^l\}_{1 \leq i \leq n}$. For any $i \in [1, n]$ and $l \in \mathbb{N}$, let σ_i^l denote the schedule date of operation instance A_i^l. Basic cyclic scheduling problems are constrained by *uniform dependences* denoted

$$A_i \xrightarrow{\theta_i^j, \omega_i^j} A_j,$$

where the *latency* θ_i^j and the *distance* ω_i^j are non-negative integers:

$$A_i \xrightarrow{\theta_i^j, \omega_i^j} A_j \Longrightarrow \sigma_i^l + \theta_i^j \leq \sigma_j^{l+\omega_i^j} \quad \forall l \in \mathbb{N} \tag{18.1}$$

In case of *cyclic scheduling on parallel processors* [HAN 95], basic cyclic scheduling problems are augmented with resource constraints. Each generic operation A_i has a processing time p_i and there are m identical processors available. Each A_i^l requires one of the processors to be available between dates $[\sigma_i^l, \sigma_i^l + p_i - 1]$.

For instruction scheduling applications, we are interested in *resource-constrained cyclic scheduling problems*, whose resource constraints are adapted from the standard RCPSP. Precisely, a set of q renewable resources is defined with availabilities $B_k > 0, \forall R_k \in \mathcal{R}$. Assume Res_i^k is a set of non-negative integers such that $c \in Res_i^k$ means operation A_i^l requires b_{ik} units of resource R_k exactly c time periods after starting its execution. In RCPSP, an operation requires its resources during all its processing time and in such cases $Res_i^k = \{0, \ldots, p_i - 1\}$ for any resource R_k and any operation A_i^l. The cyclic scheduling resource constraints can be written as:

$$\forall R_k \in \mathcal{R}, \forall t \geq 0 : \sum_{A_i^l \in \mathcal{A}_{tk}} b_{ik} \leq B_k \tag{18.2}$$

where $\mathcal{A}_{tk} = \{A_i^l | \exists l \geq 0, \exists c \in Res_i^k, \sigma_i^l + c = t\}$.

18.2.2. *The Resource-constrained modulo scheduling problem*

Modulo scheduling is a specialization of cyclic scheduling where any cyclic schedule $\{\sigma_i^l\}_{1 \leq i \leq n}^{l \in \mathbf{N}}$ must be 1-periodic with integral period $\lambda \geq 1$, traditionally called the *initiation interval*. By definition, a 1-periodic cyclic schedule satisfies:

$$\forall i \in [1, n], \forall l \geq 0 : \sigma_i^l = \sigma_i^0 + l\lambda. \tag{18.3}$$

Restricting cyclic scheduling to 1-periodic schedules allows significant simplifications in the problem formulation. Let us define $\sigma_i \stackrel{\text{def}}{=} \sigma_i^0$. The values $\{\sigma_i\}$ can be considered as the schedule dates of the generic operations $\{A_i\}$. A solution is fully characterized by the initiation interval λ and the generic schedule $\{\sigma_i\}$. Each uniform dependence constraint becomes:

$$\sigma_i + \theta_i^j - \omega_i^j \lambda \leq \sigma_j \quad \forall (i, j) \in E_{dep}. \tag{18.4}$$

For resource constraints (18.2), the definition of $\mathcal{A}_{tk}$ becomes $\mathcal{A}_{tk} = \{A_i | \exists c \in Res_i^k, \sigma_i = (t - c) \text{mod} \lambda\}$. It follows from this definition that all resource constraints (18.2) such that $t \geq \lambda$ are redundant. Hence the number of resource constraints can be reduced. They are now defined as:

$$\sum_{A_i \in \mathcal{A}_{tk}} b_{ik} \leq B_k \quad \forall R_k \in \mathcal{R}, \forall t \in [0, \lambda - 1]. \tag{18.5}$$

The primary objective of modulo scheduling is to find a 1-periodic schedule with a minimal period λ, in order to maximize the throughput. A secondary objective of modulo scheduling is often considered, such as minimizing the generic schedule length $C_{\max} = \max_{i=1,\ldots,n} \sigma_i + p_i$ or minimizing the cumulative lifetime of registers [Dup 94, Dup 96, EIC 97, Dup 04]. Both objectives are linear with the generic schedule dates so we take as the secondary objective of modulo scheduling some linear expression $\sum w_i \sigma_i$. Moreover, we assume the existence of modulo scheduling relaxations that provide deadlines d_i for each generic operation A_i, so we will include constraints $\sigma_i + p_i \leq d_i$ in the integer programming formulations discussed here.

In the traditional modulo scheduling framework [RAU 96], two lower bounds on the initiation interval λ are computed: $\lambda_{\rm rec}$ is the minimum λ such that there are no positive length circuits in the dependence graph; $\lambda_{\rm res}$ is the minimum λ such that the renewable resources $\mathcal{R}$ are not over-subscribed:

$$\lambda_{\rm rec} \stackrel{\rm def}{=} \max_C \left\lceil \frac{\sum_C \theta_i^j}{\sum_C \omega_i^j} \right\rceil : \quad C \text{ dependence circuit}$$
$$\lambda_{\rm res} \stackrel{\rm def}{=} \max_{1 \leq k \leq q} \left\lceil \frac{\sum_{i=1}^n p_i b_{ik}}{B_k} \right\rceil : \quad q = |\mathcal{R}|$$

Given $\lambda_{\min} \stackrel{\rm def}{=} \max(\lambda_{\rm rec}, \lambda_{\rm res})$, the classic modulo scheduling framework finds a minimal value of the period λ by dichotomy search such that (18.4–18.5) are heuristically feasible. Given a feasible λ, the heuristic modulo scheduler seeks to minimize the secondary objective. In accordance with this strategy, we assume there exists a higher-level search for the minimum λ such that (18.4-18.5) are feasible. At each step of this process, the value of λ is constant so we shall consider the problem of minimizing some $\sum w_i \sigma_i$ under constraints $\sigma_i + p_i \leq d_i$ given a fixed λ.

18.2.3. *From modulo scheduling to software pipelining*

To illustrate the resource-constrained modulo scheduling problems that occur in software pipelining, a sample C program and the corresponding ST200 VLIW processor operations are given in Figure 18.1. The operations are numbered in their appearance order. The ST200 processor executes up to four operations per time unit with a maximum of one control operation (goto, jump, call, return), one memory operation (load, store, prefetch), and two multiply operations per time unit. All arithmetic operations operate on integer values with operands belonging either to the General Register file (64 $\times$ 32-bit) or to the Branch Register file (8 $\times$ 1-bit). In order to eliminate some conditional branches, the ST200 VLIW architecture also provides conditional selection instructions. The processing time of any operation is a single time unit ($p_i = 1$), while the latencies θ_i^j between operations range from 0 to 3 time units.

The resource availabilities and the resource requirements of each operation are displayed in Table 18.1. The resources are: Issue for the instruction issue width; Memory for the memory access unit; Control for the control unit. An artificial resource Align is also introduced to satisfy some encoding constraints. Operations with identical resource requirements are factored into *classes*: ALU, MUL, MEM and CTL correspond respectively to the arithmetic, multiply, memory and control operations. The classes ALUX, MULX and MEMX represent the operations that require an extended immediate operand. Operations named `LDH`, `MULL`, `ADD`, `CMPNE` and `BRF` belong respectively to classes MEM, MUL, ALU, ALU and CTL.

```
int                                    L?__0_8:
prod(int n, short a[], short b) {      LDH_1 g131 = 0, G127
   int s=0, i;                         MULL_2 g132 = G126, g131
   for (i=0;i<n;i++) {                 ADD_3 G129 = G129, g132
      s += a[i]*b;                     ADD_4 G128 = G128, 1
   }                                   ADD_5 G127 = G127, 2
   return s;                           CMPNE_6 b135 = G118, G128
}                                      BRF_7 b135, L?__0_8
```

Figure 18.1. *A sample C program and the corresponding ST200 operations*

Resource	Issue	Memory	Control	Align
Availability	4	1	1	2
ALU	1	0	0	0
ALUX	2	0	0	1
MUL	1	0	0	1
MULX	2	0	0	1
MEM	1	1	0	0
MEMX	2	1	0	1
CTL	1	0	1	1

Table 18.1. *Resource availabilities and operation requirements for the ST200 VLIW processor*

The dependence graph corresponding to the example in Figure 18.1 is displayed in Figure 18.2, where each node represents a (generic) operation and each arc represents a (uniform) dependence. A dummy node 0 represents the start of the schedule and a dummy node $n+1=8$ represents the end of the schedule. Pairs of values for (θ_i^j, ω_i^j) are displayed close to their arc. Figure 18.3 displays the non-redundant dependences when $\lambda = 2$ with the values $\theta_i^j - \omega_i^j \lambda$ close to their arc. On this graph, the longest path from node 0 to node i gives the start time σ_i ignoring the resource constraints. The longest path from node 0 to node $n+1$ represents the makespan of the local schedule still ignoring the resource constraints.

A feasible modulo schedule with $\lambda = 2$ and $C_{\max} = 6$ is displayed in Figure 18.4, where each operation is suffixed by the instance number. The operations of the local schedule (instance 0) are highlighted. When the code for the resulting software pipelined loop is generated, the cyclic schedule is actually folded back into a new loop. On this example, the branch operation `BRF_7.1` would branch back to time

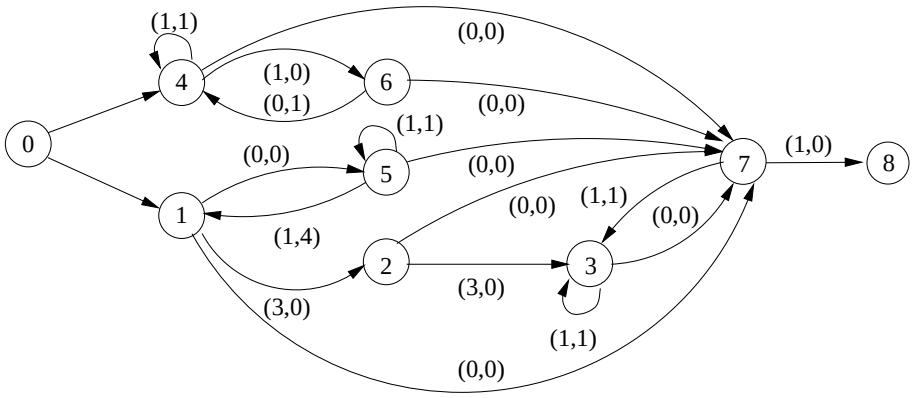


Figure 18.2. *Generic dependences of the modulo scheduling problem*

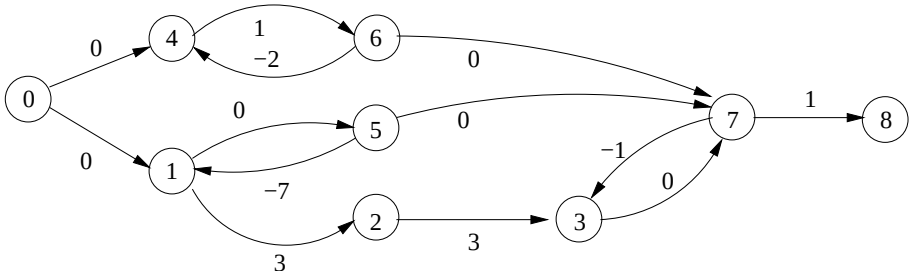


Figure 18.3. *Dependences of the modulo scheduling problem for* $\lambda = 2$

step 5 while the branch operation `BRF_7.0` would be inverted to exit from the software pipelined loop. This new loop would execute two iterations every 4 time units, yielding an effective throughput of one iteration per two time units.

The reason for generating code this way, as opposed to operation `BRF_7.0` looping back to time step 5, is to prevent loop temporary values in registers from being overwritten by the next iteration before they are no longer needed. For instance, variable `g132` is produced at time step 3 by operation `2.0` and is consumed at time step 6 by operation `3.0`. Operation `2.1`, which is executed at time step 5, must be modified to write into another variable for use at time step 8 by operation `3.1`. These code generation techniques, called *kernel unrolling* and *modulo expansion* [Dup 04], are applied after modulo scheduling and do not interfere with the scheduling problem itself.

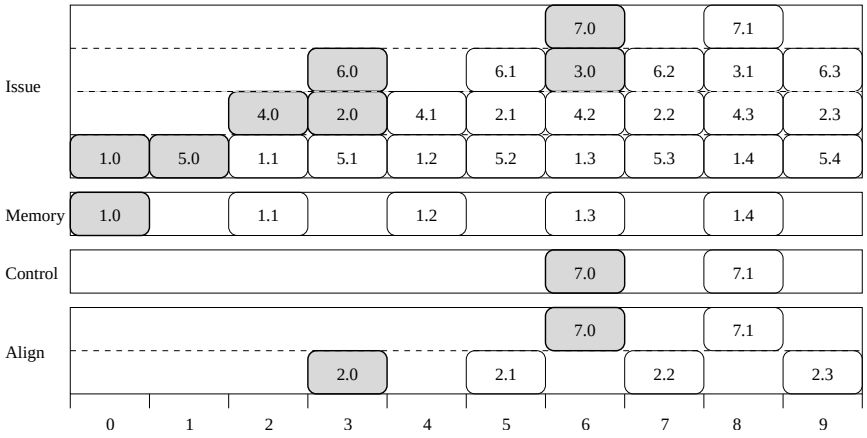


Figure 18.4. *A 1-periodic schedule with period* $\lambda = 2$ *and makespan* $C_{\max} = 6$

18.3. Integer linear programming formulations

18.3.1. *The RCMSP formulations by Eichenberger et al.*

Eichenberger *et al.* [EIC 95, EIC 97] introduce a class of integer linear programming formulations for the RCMSP, based on the decomposition of the generic schedule dates $\{\sigma_i\}$ into a quotient and a remainder:

$$\sigma_i = \lambda l_i + \tau_i : l_i \in \mathbb{N} \wedge \tau_i \in [0, \lambda - 1].$$

For the integer linear programming formulations, 0-1 time indexed variables y_i^τ are used to represent the remainders $\tau_i = \sigma_i \mathrm{mod} \lambda$, i.e. $y_i^\tau = 1 \Longleftrightarrow \tau_i = \tau$, while the quotients $l_i = \lfloor \frac{\sigma_i}{\lambda} \rfloor$ are directly defined as integer variables:

$$(ED) \qquad \sum_{i \in [1,n]} w_i (\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + l_i \lambda) : \qquad \text{minimize}$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1 \qquad i \in [1,n] \tag{18.6}$$

$$\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + l_i \lambda + \theta_i^j - \lambda \omega_i^j \leq \sum_{\tau=0}^{\lambda-1} \tau y_j^\tau + l_j \lambda \qquad (i,j) \in E_{dep} \tag{18.7}$$

$$\sum_{i=1}^{n} \sum_{c \in Res_i^k} y_i^{(\tau-c) \bmod \lambda} b_{ik} \leq B_k \qquad \tau \in [0, \lambda-1], R_k \in \mathcal{R} \tag{18.8}$$

$$l_i \in \{0, .., \lfloor \frac{d_i - p_i}{\lambda} \rfloor\} \qquad i \in [1, n] \tag{18.9}$$

$$y_i^\tau \in \{0, 1\} \qquad i \in [1, n], \tau \in [0, \lambda-1] \tag{18.10}$$

Eichenberger and Davidson [EIC 97] report that constraints (18.6–18.7) were initially proposed by Govindarajan *et al.* [GOV 94]. Constraints (18.6) state that each generic operation has to be started exactly once in time window $[0, \lambda-1]$. We have $\sigma_i = \lambda l_i + \tau_i$ and $\tau_i = \sum_{\tau=0}^{\lambda-1} \tau y_i^\tau$, so constraints (18.7) correspond exactly to constraints (18.4). Likewise, the economic function $\sum_{i \in [1,n]} w_i \sigma_i$ becomes $\sum_{i \in [1,n]} w_i (\sum_{\tau=0}^{\lambda-1} \tau y_i^\tau + l_i \lambda)$. Resource constraints (18.8) with $b_{ik} = 1$ were introduced in [EIC 95] as a linear form of constraints (18.5). Note that we allow integer values for b_{ik}, making the problem considered by [EIC 97] a particular case of the RCMSP considered here. Conversely, we do not include the register pressure equations of [EIC 95] in the above formulation, as they correspond to a specific case where the secondary objective of modulo scheduling is not linear with the generic schedule dates.

Based on results obtained by Chaudhuri *et al.* [CHA 94], Eichenberger and Davidson [EIC 97] propose a stronger formulation of the dependence constraints given by:

$$\sum_{\tau=t}^{\tau=\lambda-1} y_i^\tau + \sum_{\tau=0}^{(t+\theta_i^j-1) \bmod \lambda} y_j^\tau + l_i - l_j \leq \omega_i^j - \lfloor \frac{t + \theta_i^j - 1}{\lambda} \rfloor + 1 \qquad t \in [0, \lambda-1], (i,j) \in E_{dep} \tag{18.11}$$

By replacing constraints (18.7) by (18.11) in (ED), one obtains formulation $(ED+)$.

18.3.2. *A new time-indexed RCMSP formulation*

We propose a new formulation for the RCMSP, based on the remark that assuming a constant λ transforms the problem into a RCPSP with minimal and maximal time

lags (see Chapter 11 and [BAR 88]) and time-dependent resource requirements. A general non-preemptive time-indexed formulation has been proposed for such problem by Pritsker *et al.* [PRI 69]. This formulation is based on 0-1 variables x_i^t such that $\sigma_i = \sum_{i=0}^{d_i-p_i} t x_i^t$. By inserting λ and the resource constraints given by the Res_i^k time sets, we obtain the following integer linear programming (ILP) formulation:

$$(F) \qquad \sum_{i\in[1,n]} w_i\Big(\sum_{t=0}^{d_i-p_i} t x_i^t\Big): \qquad \text{minimize}$$

$$\sum_{t=0}^{d_i-p_i} x_i^t = 1 \qquad i \in [1,n] \tag{18.12}$$

$$\sum_{t=0}^{d_i-p_i} t x_i^t + \theta_i^j - \lambda\omega_i^j \leq \sum_{t=0}^{d_j-p_j} t x_j^t \qquad (i,j) \in E_{dep} \tag{18.13}$$

$$\sum_{i=1}^{n} \sum_{l=0}^{\lfloor \frac{d_i-p_i}{\lambda} \rfloor} \sum_{\substack{c\in Res_i^k,\\ \tau+l\lambda-c\geq 0}} x_i^{\tau+l\lambda-c} b_{ik} \leq B_k \qquad \tau \in [0,\lambda-1], R_k \in \mathcal{R} \tag{18.14}$$

$$x_i^t \in \{0,1\} \qquad i \in [1,n], t \in [0, d_i - p_i] \tag{18.15}$$

For the time-indexed RCPSP formulations, Christofides *et al.* [CHR 87] introduced disaggregated dependence constraints. Adapting these to the time lags $\theta_i^j - \lambda\omega_i^j$ yields:

$$\sum_{s=t}^{d_i-p_i} x_i^s + \sum_{s=0}^{t+\theta_i^j-\lambda\omega_i^j-1} x_j^s \leq 1 \quad t \in [0, d_i - p_i], (i,j) \in E_{dep} \tag{18.16}$$

The formulation $(F+)$ is obtained by replacing (18.13) by (18.16) in (F).

18.4. Solving the RCMSP formulations

18.4.1. *Reducing the size of the RCMSP formulations*

In terms of variable numbers, formulations (ED) and $(ED+)$ comprise $n\lambda$ binary variables y_i^τ and n integer variables l_i. Formulations (F) and $(F+)$ comprise

$\sum_{i=1}^{n}(d_i - p_i + 1)$ binary variables. Since we can assume that $d_i \geq \lambda$, the new formulation has potentially a greater number of variables.

Formulation $(F+)$ involves variables and constraints in numbers that are directly related to any assumed earliest $\{ES_i\}_{1\leq i\leq n}$ and latest $\{LS_i\}_{1\leq i\leq n}$ schedule dates (*margins*). Indeed, the number of variables is $\sum_{i=1}^{n} LS_i - ES_i + 1$. Most of the constraints are the dependence equations (18.16), and $e \stackrel{\text{def}}{=} |E_{dep}|$ non transitively redundant dependences appear to generate eT equations with $T \stackrel{\text{def}}{=} \max_{1\leq i\leq n} LS_i + p_i$. However, the first sum of (18.16) is 0 if $t > LS_i$. Likewise, the second sum of (18.16) is 0 if $t + \theta_i^j - \lambda\omega_i^j - 1 < ES_j$. (18.16) is actually equivalent to:

$$\sum_{s=t}^{d_i-p_i} x_i^s + \sum_{s=0}^{t+\theta_i^j-\lambda\omega_i^j-1} x_j^s \leq 1 \qquad t \in [ES_j - \theta_i^j + \lambda\omega_i^j + 1, LS_i], (i,j) \in E_{dep} \tag{18.17}$$

Therefore, (18.17) is redundant whenever $ES_j - \theta_i^j + \lambda\omega_i^j \geq LS_i$.

Due to the decrease in the number for variables and equations, formulation $(F+)$ is likely to become more tractable after reducing the assumed margins. By comparison, the number of variables in formulation $(ED+)$ only decreases when some $LS_i - ES_i \leq \lambda$, while most dependence equations (18.11) remain.

For the RCPSP, the basic technique to reduce the margins is to initialize $ES_i = r_i$ and $LS_i = d_i - p_i$, with $\{r_i\}_{1\leq i\leq n}$ and $\{d_i\}_{1\leq i\leq n}$ being the release dates and the deadlines, then propagate the dependence constraints using a label-correcting algorithm. More elaborate constraint propagation techniques have been proposed for the RCPSP (see Chapter 4 and [DEM 05]). Ultimately, all these margins reduction techniques rely on some effective upper bounding of the deadlines $\{d_i\}_{1\leq i\leq n}$.

18.4.2. *A large neighborhood search for the RCMSP*

In the case of the modulo scheduling, the primary objective is the period minimization. Heuristics build modulo schedules at some period λ that is often greater than the lower bound $\lambda_{\min} \stackrel{\text{def}}{=} \max(\lambda_{\text{rec}}, \lambda_{\text{res}})$. When this happens, the feasibility of modulo scheduling at period $\lambda - 1$ is open. Moreover, it is not known how to bound the makespan at period $\lambda - 1$ given a makespan at period λ. This means that effective upper bounding of the due dates in RCMSP instances is not available either.

To compensate for this lack of upper bounding, we propose a large neighborhood search (LNS) for the RCMSP formulations, based on adaptive margins reduction and implicit enumeration of the resulting time-indexed integer programs (using a MIP

solver). The LNS (see [PAL 04] and Chapters 6 and 7) is a meta-heuristic where a large number of solutions in the neighborhood of an incumbent solution are searched by means of branch-and-bound , constraint programming, or integer programming. The large neighborhood is obtained by fixing some variables of the incumbent solution while releasing others.

In the setting of time-indexed formulations, we consider as the neighborhood of an incumbent solution $\{\sigma_i\}_{1\leq i\leq n}$ some margins $\{ES_i, LS_i : \sigma_i \in [ES_i, LS_i]\}_{1\leq i\leq n}$, which are made consistent under dependence constraint propagation. For each $x_i^{\sigma_i} = 1$, we fix variables $x_i^{r_i} \ldots x_i^{ES_i-1}$, $x_i^{LS_i+1} \ldots x_i^{d_i-p_i}$ to zero and release variables $x_i^{ES_i} \ldots x_i^{LS_i}$. The fixed variables and the dependence constraints (18.17) found redundant given the margins are removed from the integer program.

We adapt the generic LNS algorithm of [PAL 04] by using margins to define the neighborhoods and by starting from a heuristic modulo schedule. The period λ is kept constant while our LNS searches increasingly wider margins under a time budget in order to minimize the makespan. The key innovation is to replace the diversification operator of [PAL 04] by a decrement of the period to $\lambda-1$ while keeping the schedule dates of the former period λ. This yields a pseudo-solution that may no longer be feasible due to the period change. Computational experience shows that a new solution at period $\lambda-1$ can often be found in the neighborhood of this pseudo-solution, whenever the problem instance is feasible at period $\lambda-1$. In the case a solution at period $\lambda-1$ is found, our LNS goes back to the makespan minimization.

18.4.3. *Implementation and Experiments*

We implemented the two time-indexed formulations of RCMSP $(ED+)$ and $(F+)$ in the production compiler for the STMicroelectronics ST200 VLIW processor family [Dup 04], along with the proposed LNS heuristic, then used the CPLEX 9.0 Callable Library to solve the resulting MIPs.

Table 18.2 reports LNS experimental data for the largest loops that could not be directly solved with these formulations, assuming a timeout of 300 s and a time horizon of $4\lambda_{\min}$. Column *#O,#D* gives the number of operations and of non-redundant dependences. Column *Heuristic* λ, M displays the period and makespan computed by the ST200 production compiler insertion scheduling heuristic [Dup 04]. The column groups $(F+)$ *300*, $(F+)$ *30*, $(ED+)$ *300*, correspond to the proposed LNS using our formulation and Eichenberger *et al* formulation for timeout values of 300 s, 30 s, 300 s. In each group, column *#V,#C* gives the number of variables and constraints of the integer program sent to CPLEX 9.0 by the last successful LNS step. In all these cases, the $(F+)$ LNS reached the $\lambda_{\min}$, thus ensuring optimality of the period λ.

Loop	#O,#D	Heuristic	$(F+)$ 300		$(F+)$ 30		$(ED+)$ 300	
		λ, M	λ, M	#V,#C	λ, M	#V,#C	λ, M	#V,#C
q_plsf_5.0_215	231,313	81,97	75,77	1114,1236	75,78	1873,2042	*,*	18942,25989
q_plsf_5.0_227	121,163	42,92	39,46	982,1228	39,46	1378,1685	39,47	4840,6673
q_plsf_5.0_201	124,168	42,92	40,47	1086,1340	40,65	1197,1421	41,50	5208,7217
q_plsf_5.2_11	233,317	82,100	75,78	1113,1216	75,79	1897,2045	*,*	19339,26637
subbands.0_196	130,234	44,65	35,49	718,906	35,48	1008,1248	*,*	5850,10778
transfo.IMDCT_L	232,370	71,109	58,58	1133,1075	58,58	1985,1961	70,74	16472,26482

Table 18.2. *Large-scale neighborhood search experimental data.*

18.5. Summary and conclusions

High-quality instruction scheduling for VLIW processors, in particular the software pipelining of inner loops, is a significant contributor to the performance of media processing applications. Modern software pipelining techniques operate under the framework of modulo scheduling [RAU 96], which simplifies the cyclic scheduling problems by focusing on 1-periodic cyclic schedules with integral period. In addition to minimizing the period, modulo schedulers seeks to reduce a secondary objective that can be expressed as a weighted linear sum of the generic schedule dates.

In this chapter, we formalize the resource-constrained modulo scheduling problem and we focus on two integer programming formulations of this problem. The first formulation is based on work by Eichenberger *et al.* [EIC 95, EIC 97]. The second formulation is generalized from the classic non-preemptive time-indexed integer programming formulation of Pritsker *et al* [PRI 69] for the RCPSP (RCPSP) with time-lags. For both formulations, we consider the variants where the dependence constraints are disaggregated.

Our computational experience shows that, while the formulation of Eichenberger *et al.* generates fewer variables than our formulation generalized from Pritsker *et al.*, both techniques are intractable in practice for modulo scheduling problems with more than a few dozen generic operations. However, when combined with a simple large-scale neighborhood search, only the new formulation with disaggregated dependence equations makes it possible to solve near-optimally modulo scheduling problems with hundreds of generic operations and uniform dependences in a matter of minutes.

Bibliography

[ADE 02] ADELSON-VELSKY G. M., LEVNER E., "Project scheduling in AND-OR graphs: A generalization of Dijkstra's algorithm", *Mathematics of Operations Research*, vol. 27, num. 3, p. 504-517, 2002.

[AHU 76] AHUJA H., *Construction Performance Control by Networks*, Wiley & Sons, 1976.

[AKT 99] AKTURK M., GORGULU E., "Match-up scheduling under a machine breakdown", *European Journal of Operational Research*, vol. 112, num. 1, p. 81–97, 1999.

[ALF 97] ALFARES H. K., BAILEY J., "Integrated project task and manpower scheduling", *IIE Transactions*, vol. 29, num. 9, p. 711–717, 1997.

[ALL 95] ALLAN V. H., JONES R. B., LEE R. M., ALLAN S. J., "Software pipelining", *ACM Computing Surveys*, vol. 27, num. 3, p. 367–432, 1995.

[ALV 89] ALVAREZ-VALDÉS R., TAMARIT J. M., "Heuristic algorithms for Resource-Constrained Project Scheduling: a review and an empirical analysis", SLOWINSKI R., WEGLARZ J., Eds., *Advances in Project Scheduling*, p. 113-134, Elsevier, 1989.

[ALV 93] ALVAREZ-VALDÉS R., TAMARIT J. M., "The project scheduling polyhedron: dimension, facets and lifting theorems", *European Journal of Operational Research*, vol. 67, num. 2, p. 204–220, 1993.

[APP 91] APPLEGATE D., COOK W., "A computational study of job-shop scheduling", *ORSA Journal on Computing*, vol. 3, num. 2, p. 149–156, 1991.

[ART 00] ARTIGUES C., ROUBELLAT F., "A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes", *European Journal of Operational Research*, vol. 127, num. 2, p. 297-316, 2000.

[ART 03] ARTIGUES C., MICHELON P., REUSSER S., "Insertion techniques for static and dynamic Resource-Constrained Project Scheduling", *European Journal of Operational Research*, vol. 149, num. 2, p. 249-267, 2003.

[AYT 05] AYTUG H., LAWLEY M., MCKAY K., MOHAN S., UZSOY R., "Executing production schedules in the face of uncertainties: A review and some future directions", *European Journal of Operational Research*, vol. 161, num. 1, p. 86–110, 2005.

[BAA 98] BAAR T., BRUCKER P., KNUST S., "Tabu-search algorithms and lower bounds for the Resource-Constrained Project Scheduling Problem", VOSS S., MARTELLO S., OSMAN I., ROUCAIROL C., Eds., *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, p. 1-18, Kluwer Academic Publishers, 1998.

[BAI 95] BAILEY J., ALFARES H. K., LIN W. Y., "Optimization and heuristic models to integrate project task and manpower scheduling", *Computers and Industrial Engineering*, vol. 29, num. 1, p. 473–376, 1995.

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

[BAK 94] BAKER A. B., "The hazards of fancy backtracking", *Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA, USA, AAAI Press, p. 288-293, 1994.

[BAL 70] BALAS E., "Project scheduling with resource constraints", BEALE E. M. L., Ed., *Applications of Mathematical Programming Techniques*, p. 187–200, American Elsevier, 1970.

[BAL 79] BALAS E., "Disjunctive programming", *Annals of Discrete Mathematics*, vol. 5, p. 3–51, 1979.

[BAL 85] BALAS E., "On the facial structure of scheduling polyhedra", *Mathematical Programming Study*, vol. 24, p. 179–218, 1985.

[BAL 07] BALLESTÍN F., TRAUTMANN N., "An iterated-local-search heuristic for the resource-constrained weighted earliness-tardiness project scheduling problem", *International Journal of Production Research*, 2007, to appear.

[BAN 94] BANDELLONI M., TUCCI M., RINALDI R., "Optimal resource leveling using non-serial dynamic programming", *European Journal of Operational Research*, vol. 78, num. 2, p. 162–177, 1994.

[BAP 98] BAPTISTE P., Une étude théorique et expérimentale de la propagation de contraintes de ressources, PhD thesis, Université de Technologie de Compiègne (France), 1998.

[BAP 99] BAPTISTE P., LE PAPE C., NUIJTEN W., "Satisfiability tests and time-bound adjustments for cumulative scheduling problems", *Annals of Operations Research*, vol. 92, p. 305–333, 1999.

[BAP 00] BAPTISTE P., LE PAPE C., "Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems", *Constraints*, vol. 5, num. 1–2, p. 119–139, 2000.

[BAP 01] BAPTISTE P., LE PAPE C., NUIJTEN W., *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, Kluwer Academic Publishers, 2001.

[BAP 04] BAPTISTE P., DEMASSEY S., "Tight LP bounds for Resource Constrained Project Scheduling", *OR Spectrum*, vol. 26, num. 2, p. 251 - 262, 2004.

[BAP 05] BAPTISTE P., CARLIER J., KONONOV A., QUEYRANNE M., SEVASTIANOV S., SVIRIDENKO M., "Structural properties of preemptive schedules", submitted for publication, 2005.

[BAP 06] BAPTISTE P., LABORIE P., LE PAPE C., NUIJTEN W., "Constraint-based scheduling and planning", F. ROSSI P. VAN BEEK T. W., Ed., *Handbook of Constraint Programming*, p. 759-798, Elsevier, 2006.

[BAR 88] BARTUSCH M., MÖHRING R. H., RADERMACHER F. J., "Scheduling project networks with resource constraints time windows", *Annals of Operations Research*, vol. 16, p. 201–240, 1988.

[BAR 99] BAROUM S., PATTERSON J., "An exact solution procedure for maximizing the net present value of cash flows in a network", WEGLARZ J., Ed., *Handbook on Recent Advances in Project Scheduling*, p. 107-134, Kluwer Academic Publishers, 1999.

[BEA 91] BEAN J., BIRGE J., MITTENTHAL J., NOON C., "Match-up scheduling with multiple resources, release dates and disruptions", *Operations Research*, vol. 39, num. 3, p. 470–483, 1991.

[BEA 01] BEAUSEIGNEUR M., BOUTEVIN C., GOURGAND M., NORRE S., "A generic simulation model for an industrial problem", GIAMBIASI N., FRYDMAN C., Eds., *13th European Simulation Symposium (ESS 2001)*, Marseille, France, p. 182 - 186, 2001.

[BEI 92] BEIN W., KAMBUROWSKI J., STALLMANN M., "Optimal reduction of two-terminal directed acyclic graphs", *SIAM Journal on Computing*, vol. 21, num. 6, p. 1112–1129, 1992.

[BEL 91] BELL C. E., HAN J., "A new heuristic solution method in Resource-Constrained Project Scheduling", *Naval Research Logistics*, vol. 38, num. 3, p. 315–331, 1991.

[BEL 96] BELDICEANU N., BOURREAU E., RIVREAU D., SIMONIS H., "Solving Resource-Constrained Project Scheduling with CHIP", *The Fifth International Workshop on Project Management and Scheduling (PMS'1996)*, Poznań, Poland, 1996.

[BEL 05] BELLENGUEZ O., NÉRON E., "Lower bounds for the multi-skill Project Scheduling Problem with hierarchical levels of skills", BURKE E. K., TRICK M., Eds., *Practice and Theory of Automated Timetabling V*, vol. 3616 of *Lecture Notes in Computer Science*, Springer-Verlag, p. 229–243, 2005.

[BEL 07] BELLENGUEZ-MORINEAU O., E.NÉRON, "A branch-and-bound method for solving multi-skill project scheduling problem", *RAIRO - Operations Research*, vol. 41, num. 2, p. 155–170, 2007.

[BEN 97] BENDALI F., QUILLIOT A., "Representation of ordered families of intervals, and applications", *RAIRO - Operations Research*, vol. 31, num. 1, p. 73–101, 1997.

[BES 06] BESSIÈRE C., "Constraint propagation", F. ROSSI P. VAN BEEK T. W., Ed., *Handbook of Constraint Programming*, p. 27-82, Elsevier, 2006.

[BIE 99] BIERWIRTH C., MATTFELD D., "Production scheduling and rescheduling with genetic algorithms", *Evolutionary Computation*, vol. 7, num. 1, p. 1–17, 1999.

[BIL 05] BILLAUT J.-C., MOUKRIM A., SANLAVILLE E., Eds., *Flexibilité et Robustesse en Ordonnancement*, Hermès, 2005.

[BLA 83] BLAZEWICZ J., LENSTRA J., KAN A. R., "Scheduling subject to resource constraints: Classification and complexity", *Discrete Applied Mathematics*, vol. 5, num. 1, p. 11-24, 1983.

[BOC 96] BOCTOR F., "Resource-Constrained Project Scheduling by simulated annealing", *International Journal of Production Research*, vol. 34, num. 8, p. 2335-2351, 1996.

[BOU 01] BOUTEVIN C., GOURGAND M., NORRE S., "Coupling heuristics, metaheuristics and simulation model for an industrial problem similar to RCPSP", *International conference on industrial engineering and production management (IEPM'2001)*, Québec, Canada, 2001.

[BOU 03] BOULEIMEN K., LECOCQ H., "A new efficient simulated annealing algorithm for the Resource-Constrained Project Scheduling Problem and its multiple mode version", *European Journal of Operational Research*, vol. 149, num. 2, p. 268-281, 2003.

[BOU 04] BOULY H., Généralisation d'algorithmes d'ordonnancement à la production de ressources par les tâches, Master's thesis, Université de Technologie de Compiègne - Ecole Supérieure d'Ingénieurs en Electrotechnique et Electronique d'Amiens, 2004.

[BRI 96] BRINKMANN K., NEUMANN K., "Heuristic procedures for Resource-Constrained Project Scheduling with minimal and maximal time lags: the resource-leveling and minimum project duration problems", *Journal of Decision Systems*, vol. 5, p. 129–155, 1996.

[BRI 06] BRIAND C., BEZANGER J., "An any-order SGS for project scheduling with scarce resources and precedence constraints", *10th International Workshop on Project Management and Scheduling (PMS'2006)*, Poznan, Poland, p. 95–100, 2006.

[BRO 65] BROOKS G., WHITE C., "An algorithm for finding optimal or near optimal solutions to the production scheduling problem", *Journal of Industrial Engineering*, vol. 16, p. 34-40, 1965.

[BRU 83] BRUER H., MÖHRING R., "A fast algorithm for the decomposition of graphs and posets", *Mathematics of Operations Research*, vol. 8, p. 170–184, 1983.

[BRU 98] BRUCKER P., KNUST S., SCHOO A., THIELE O., "A branch and bound algorithm for the Resource-Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 107, p. 272–288, 1998.

[BRU 99] BRUCKER P., DREXL A., MÖHRING R., NEUMANN K., PESCH E., "Resource-constrained project scheduling: notation, classification, models, and methods", *European Journal of Operational Research*, vol. 112, num. 1, p. 3-41, 1999.

[BRU 00] BRUCKER P., KNUST S., "A linear programming and constraint propagation-based lower bound for the RCPSP", *European Journal of Operational Research*, vol. 127, p. 355–362, 2000.

[BRU 03] BRUCKER P., KNUST S., "Lower bounds for the Resource-Constrained Project Scheduling Problems", *European Journal of Operational Research*, vol. 149, p. 302–313, 2003.

[BRU 06] BRUCKER P., KNUST S., *Complex scheduling*, Springer-Verlag, 2006.

[BUR 91] BURKE P., PROSSER P., "A distributed asynchronous system for predictive and reactive scheduling", *International Journal for Artificial Intelligence in Engineering*, vol. 6, num. 3, p. 106–124, 1991.

[BUS 93] BUSS A., ROSENBLATT M., Activity delay in stochastic project networks, Report , Washington University, 1993.

[BUS 95] BUSS A., ROSENBLATT M., "Resource-Constrained scheduling in stochastic project networks", *INFORMS International Singapore Meeting*, 1995.

[CAR 82] CARLIER J., "The one-machine sequencing problem", *European Journal of Operational Research*, vol. 11, num. 1, 1982.

[CAR 84] CARLIER J., Problèmes d'ordonnancements à contraintes de ressources: algorithmes et complexité, Doctorat d'état, PhD thesis, Université Pierre et Marie Curie (Paris VI) (France), 1984.

[CAR 87] CARLIER J., "Scheduling jobs with release dates and tails on identical machines to minimize the makespan", *European Journal of Operational Research*, vol. 29, p. 298–306, 1987.

[CAR 89] CARLIER J., PINSON E., "An algorithm for solving the job-shop problem", *Management Science*, vol. 35, p. 164–176, 1989.

[CAR 91] CARLIER J., LATAPIE B., "Une méthode arborescente pour résoudre les problèmes cumulatifs", *RAIRO - Recherche Opérationnelle*, vol. 25, num. 3, p. 311–340, 1991.

[CAR 98] CARLIER J., PINSON E., "Jackson's Pseudo Preemptive Schedule for the $Pm/r_i, q_i/C_{max}$ scheduling problem", *Annals of Operations Research*, vol. 83, p. 41–48, 1998.

[CAR 03] CARLIER J., NÉRON E., "On linear lower bounds for the Resource Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 149, p. 314–324, 2003.

[CAR 04] CARLIER J., PINSON E., "Jackson's pseudo preemptive schedule and cumulative scheduling problems", *Discrete Applied Mathematics*, vol. 145, p. 80–94, 2004.

[CAR 06] CARLIER J., MOUKRIM A., XU H., "Arbitrage methods for scheduling problems with production and consumption of a non-renewable resource", *Workshop International Logistique and Transport*, Hammamet, Tunisie, p. 100-112, 2006.

[CAR 07] CARLIER J., NÉRON E., "Computing redundant resources for the Resource Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 176, p. 1452–1463, 2007.

[CAS 96] CASEAU Y., LABURTHE F., "Cumulative scheduling with task intervals", *Proceedings of the Joint International Conference and Symposium on Logic Programming*, p. 363-377, 1996.

[CAS 97] CASEAU Y., LABURTHE F., "A constraint based approach to the RCPSP", *Proceeding of the International Conference on Principles and Practice of Constraint Programming, (CP'97)*, vol. 1330 of *Lecture Notes in Computer Science*, Springer-Verlag, 1997.

[CAV 01] CAVALCANTE C., DE SOUZA C., SAVELSBERGH M., WONG Y., WOLSEY L., "Scheduling Projects with Labor Constraints", *Discrete Applied Mathematics*, vol. 112, p. 27–52, 2001.

[CES 96] CESTA A., ODDI A., "Gaining efficiency and flexibility in the simple temporal problem", *Third International Conference on Temporal Representation and Reasoning (TIME-96)*, Key-West, Florida, U.S.A., 1996.

[CHA 94] CHAUDHURI S., WALKER R. A., MITCHELL J. E., "Analysing and exploiting the structure of the constraints in the ILP approach to the scheduling problem", *IEEE Transactions on Very Large Scale Integration Systems*, vol. 2, num. 4, p. 456–471, 1994.

[CHO 97] CHO J.-H., KIM Y.-D., "A simulated annealing algorithm for the Resource-Constrained Project Scheduling Problems", *Journal of the Operational Research Society*, vol. 48, p. 736-744, 1997.

[CHR 87] CHRISTOFIDES N., ALVAREZ-VALDÉS R., TAMARIT J. M., "Project scheduling with resource constraints: a branch and bound approach", *European Journal of Operational Research*, vol. 29, num. 3, p. 262–273, 1987.

[CHU 92] CHURCH L., UZSOY R., "Analysis of periodic and event-drive rescheduling policies in dynamic shops", *International Journal of Computer Integrated Manufacturing*, vol. 5, p. 153–163, 1992.

[COL 91] COLLINOT A., LE PAPE C., "Adapting the behavior of a job-shop scheduling system", *International Journal for Decision Support Systems*, vol. 7, num. 3, p. 341–353, 1991.

[COO 76] COOPER D. F., "Heuristics for scheduling resource-constrained projects: an experimental investigation", *Management Science*, vol. 22, num. 11, p. 1186–1194, 1976.

[DAM 05] DAMAY J., Techniques de résolution basées sur la programmation linéaire pour l'ordonnancement de projet, PhD thesis, Université Blaise Pascal, Clermont-Ferrand (France), 2005.

[DAM 06a] DAMAY J., FOUILHOUX P., "Un algorithme de Branch and Cut and Price pour le RCPSP préemptif", *7ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'06)*, Lille, 2006.

[DAM 06b] DAMAY J., SANLAVILLE E., "Méthode par Séparation/Evaluation pour le RCPSP préemptif", *7ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'06)*, Lille, Presses Universitaires de Valenciennes, p. 207-218, 2006.

[DAM 08] DAMAY J., QUILLIOT A., SANLAVILLE E., "Linear programming based algorithms for preemptive and non-preemptive RCPSP", *European Journal of Operational Research, to appear*, 2008.

[DAU 96] DAUZÈRE-PÉRÈS S., ROUX W., LASSERRE J., "Multi-resource shop scheduling with resource flexibility", *European Journal of Operational Research*, vol. 107, p. 289-305, 1996.

[DAU 03] DAUZÈRE-PÉRÈS S., PAVAGEAU C., "Extension of an integrated approach for muti-resource shop flexibility", *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol. 33, num. 2, 2003.

[DAV 75] DAVIS E. W., "Project network summary measures and constrained-resource scheduling", *AIIE Transactions*, vol. 7, num. 2, p. 132–142, 1975.

[DAV 00] DAVENPORT A., BECK J., A survey of techniques for scheduling with uncertainty, http://www.eil.utoronto.ca/profiles/chris/chris.papers.html, 2000.

[DAY 93] DAYANAND N., PADMAN R., Payments in projects : a contractor's model, Report num. 93-71, Carnegie Mellon University, 1993.

[DAY 96] DAYANAND N., Scheduling payments in projects : an optimization-based approach, PhD thesis, Carnegie Mellon University (USA), 1996.

[de 96a] DE REYCK B., HERROELEN W., "On the use of the complexity index as a measure of complexity in activity networks", *European Journal of Operational Research*, vol. 91, num. 2, p. 347–366, 1996.

[de 96b] DE REYCK B., HERROELEN W., An optimal procedure for the unconstrained max-npv project scheduling problem with generalized precedence relations, Report num. 9642, Department of Applied Economics, Katholieke Universiteit Leuven, 1996.

[de 98] DE REYCK B., HERROELEN W., "An optimal procedure for the Resource-Constrained Project Scheduling Problem with discounted cash flows and generalized precedence relations", *Computers and Operations Research*, vol. 25, num. 1, p. 1–17, 1998.

[DE 95] DE P., DUNNE E., GHOSH J., WELLS C., "The discrete time-cost tradeoff problem revisited", *European Journal of Operational Research*, vol. 81, p. 225–238, 1995.

[DEB 03] DEBRUYNE R., FERRAND G., JUSSIEN N., LESAINT W., OUIS S., TESSIER A., "Correctness of Constraint Retraction Algorithms", *Sixteenth international Florida Artificial Intelligence Research Society conference (FLAIRS'03)*, St. Augustine, Florida, USA, AAAI press, p. 172–176, 2003.

[DEB 06a] DEBELS D., DE REYCK B., LEUS R., VANHOUCKE M., "A hybrid scatter search/electromagnetism meta-heuristic for project scheduling", *European Journal of Operational Research*, vol. 169, num. 2, p. 638-653, 2006.

[DEB 06b] DEBLAERE F., DEMEULEMEESTER E., HERROELEN W., DE VONDER S. V., Proactive resource allocation heuristics for robust project scheduling, Report num. KBI 0608, Department of Decision Sciences & Information Management, K.U.Leuven, Belgium, 2006.

[DEC 88] DECHTER R., DECHTER A., "Belief maintenance in dynamic constraint networks", *Proc. of the 7th National Conference on Artificial Intelligence (AAAI-88)*, St. Paul, MN, USA, p. 37-42, 1988.

[DEC 91] DECHTER R., MEIRI I., PEARL J., "Temporal constraint networks", *Artificial Intelligence*, vol. 49, num. 1-3, p. 61-96, 1991.

[DEM 92a] DEMEULEMEESTER E., Optimal algorithms for various classes of multiple resource-constrained project scheduling problems, PhD thesis, Katholieke Universiteit Leuven, Belgium, unpublished, 1992.

[DEM 92b] DEMEULEMEESTER E., HERROELEN W., "A Branch-and-Bound Procedure for the multiple-resource constrained single project scheduling problem", *Management Science*, vol. 38, num. 12, p. 1803–1818, 1992.

[DEM 95] DEMEULEMEESTER E., "Minimizing resource availability costs in time-limited project networks", *Management Science*, vol. 41, p. 1590–1598, 1995.

[DEM 96a] DEMEULEMEESTER E., HERROELEN W., "An efficient optimal solution procedure for the preemptive Resource-Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 90, num. 2, p. 334–348, 1996.

[DEM 96b] DEMEULEMEESTER E., HERROELEN W., DOMMELEN P. V., An optimal recursive search procedure for the deterministic unconstrained max-NPV project scheduling problem, Report num. 9603, Department of Applied Economics, Katholieke Universiteit Leuven, 1996.

[DEM 97] DEMEULEMEESTER E., HERROELEN W., "New benchmark results for the Resource-Constrained Project Scheduling Problem", *Management Science*, vol. 43, num. 11, p. 1485–1492, 1997.

[DEM 02a] DEMASSEY S., ARTIGUES C., MICHELON P., "A hybrid Constraint propagation-cutting plane algorithm for the RCPSP", *Proceedings of the 4th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'02)*, Le Croisic, France, p. 321–331, 2002.

[DEM 02b] DEMEULEMEESTER E., HERROELEN W., *Project scheduling - A research handbook*, vol. 49 of *International Series in Operations Research & Management Science*, Kluwer Academic Publishers, 2002.

[DEM 02c] DEMEULEMEESTER E., HERROELEN W., *Project Scheduling: a Research Handbook*, vol. 49 of *International Series in Operations Research and Management Science*, Kluwer Academic Publishers, 2002.

[DEM 05] DEMASSEY S., ARTIGUES C., MICHELON P., "Constraint propagation based cutting planes : an application to the resource-constrained project scheduling problem", *INFORMS Journal on Computing*, vol. 17, num. 1, p. 52–65, 2005.

[DOE 77] DOERSCH R., PATTERSON J., "Scheduling a project to maximize its present value : a zero-one programming approach", *Management Science*, vol. 23, p. 882–889, 1977.

[DOR 99] DORNDORF U., PHAN-HUY T., PESCH E., "A Survey of Interval Capacity Consistency Tests for Time- and Resource-Constrained Scheduling", WEGLARZ J., Ed., *Project Scheduling – Recent Models, Algorithms and Applications*, p. 213–238, Kluwer Academic Publishers, 1999.

[DOR 00] DORNDORF U., PESCH E., PHAN-HUY T., "A branch-and-bound algorithm for the Resource Constrained Project Scheduling Problem", *Mathematical Methods of Operations Research*, vol. 52, p. 413–439, 2000.

[DOR 02] DORNDORF U., *Project Scheduling with Time Windows - From Theory to Applications*, Physica-Verlag, 2002.

[DRE 02] DREZET L.-E., "Suivi et planification de projets complexes", Mémoire de DEA, Laboratoire d'Informatique, Université de Tours, France, 2002.

[DRE 03] DREZET L., TACQUARD C., "Survey of financial aspects in project scheduling", *International Conference on Industrial Engineering and Production Management (IEPM 2003)*, Porto, Portugal, 2003.

[DRE 05] DREZET L.-E., Résolution d'un problème de gestion de projets sous contraintes de ressources humaines: De l'approche prédictive à l'approche réactive, PhD thesis, Université François Rabelais, Tours (France), 2005.

[Dup 94] DUPONT DE DINECHIN B., "An Introduction to Simplex Scheduling", *IFIP WG10.3 Working Conference on Parallel Architectures and Compilation Techniques*, p. 327-330, 1994.

[Dup 96] DUPONT DE DINECHIN B., "Parametric Computation of Margins and of Minimum Cumulative Register Lifetime Dates", *Languages and Compilers for Parallel Computing*, vol. 1239/1997 of *Lecture Notes in Computer Science*, p. 231-245, 1996.

[Dup 04] DUPONT DE DINECHIN B., "From machine scheduling to VLIW instruction scheduling", *ST Journal of Research*, vol. 1, num. 2, 2004.

[DYE 90] DYER M. E., WOLSEY L. A., "Formulating the single machine sequencing problem with release dates as a mixed integer program", *Discrete Applied Mathematics*, vol. 26, p. 255–270, 1990.

[EAS 89] EASA S., "Resource leveling in construction by optimisation", *Journal of Construction and Management*, vol. 115, p. 302–316, 1989.

[EIC 95] EICHENBERGER A. E., DAVIDSON E. S., ABRAHAM S. G., "Optimum modulo schedules for minimum register requirements", *International Conference on Supercomputing Proceedings (ICS'95)*, Barcelona, Spain, p. 31–40, 1995.

[EIC 97] EICHENBERGER A. E., DAVIDSON E. S., "Efficient formulation for optimal modulo schedulers", *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Las Vegas, Nevada, p. 194-205, 1997.

[ELK 03a] ELKHYARI A., Outils d'aide à la décision pour des problèmes d'ordonnancement dynamiques, PhD thesis, Université de Nantes, Nantes (France), 2003.

[ELK 03b] ELKHYARI A., GUÉRET C., JUSSIEN N., "Solving dynamic timetabling problems as dynamic resource constrained project scheduling problems using new constraint programming tools", BURKE E. K., CAUSMAECKER P. D., Eds., *Practice and Theory of Automated Timetabling IV*, num. 2740Lecture Notes in Computer Science, Springer-Verlag, p. 39–59, 2003.

[ELM 80] ELMAGHRABY S. E., HERROELEN W., "On the measurment of complexity in activity networks", *European Journal of Operational Research*, vol. 5, num. 4, p. 223–234, 1980.

[ELM 90] ELMAGHRABY S., HERROELEN W., "The scheduling of activities to maximize the net present value of projects", *European Journal of Operational Research*, vol. 49, p. 35–49, 1990.

[ELM 00] ELMAGHRABY S., FERREIRA A., TAVARES L., "Optimal start times under stochastic activity durations", *International Journal of Production Economics*, vol. 64, p. 153–164, 2000.

[ERE 93] ERENGUC S., TUFEKCI S., ZAPPE C., "The solution of the time/cost trade-off problem with discounted cash flows using generalized Benders decomposition", *Naval Research Logistics*, vol. 40, p. 25–50, 1993.

[ERS 91] ERSCHLER J., LOPEZ P., THURIOT C., "Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement", *Revue d'Intelligence Artificielle*, vol. 5, p. 7–32, 1991.

[ETG 96] ETGAR R., SHTUB A., LEBLANC L., "Scheduling projects to maximize net present value - the case of time-dependent, contingent cash flows", *European Journal of Operational Research*, vol. 96, p. 90–96, 1996.

[FAR 00] FARABOSCHI P., BROWN G., FISHER J. A., DESOLI G., HOMEWOOD F., "Lx: a technology platform for customizable VLIW embedded processing", *The 27th Annual International Symposium on Computer architecture*, New York, NY, USA, ACM Press, p. 203–213, 2000.

[FEI 98] FEIGE U., KILIAN J., "Zero-knowledge and the chromatic number", *Journal of Computer and System Sciences*, vol. 187, num. 57, p. 187-199, 1998.

[FEK 98] FEKETE S. P., SCHEPERS J., "New classes of lower bounds for bin packing problems", *Lecture Notes in Computer Science*, vol. 1412, p. 257–270, 1998.

[FIS 73] FISHER M., "Optimal solution of scheduling problems using lagrange multipliers, Part I", *Operations Research*, vol. 21, num. 5, 1973.

[FLE 85] FLEISCHMANN B., JESS H., "Erfahrungen mit einem Simulationsmodell in einer Aluminiumgießerei", *OR Spektrum*, vol. 7, p. 175–185, 1985.

[FLE 04] FLESZAR K., HINDI K., "Solving the Resource-Constrained Project Scheduling Problem by a variable neighborhood search", *European Journal of Operational Research*, vol. 155, p. 402-413, 2004.

[FLO 62] FLOYD R. W., "Algorithm 97: Shortest path", *Communications of the ACM*, vol. 5, num. 6, p. 345–345, 1962.

[FOR 97] FORTEMPS P., HAPKE M., "On the disjunctive graph for project scheduling", *Foundations on Computing and Decision Sciences*, vol. 22, p. 195–209, 1997.

[GAR 75] GAREY M., JOHNSON D., "Complexity results for multiprocessor scheduling under resource constraints", *SIAM Journal on Computing*, vol. 4, num. 4, p. 397-411, 1975.

[GAR 79] GAREY M. R., JOHNSON D. S., *Computers and Intractibility. A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

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

[GLO 99] GLOVER F., "Scatter search and path relinking", CORNE D., DORIGO M., GLOVER F., Eds., *New Ideas in Optimization*, p. 297-316, McGraw Hill, 1999.

[GOD 05] GODARD D., LABORIE P., NUIJTEN W., "Randomized large neighborhood search for cumulative scheduling", *Proceedings International Conference on Automated Planning and Scheduling (ICAPS-05)*, Monterey, California, USA, 2005.

[GON 84] GONDRAN M., MINOUX M., *Graphs and Algorithms*, John Wiley & Sons, 1984.

[GOU 91] GOURGAND M., KELLERT P., "Conception d'un environnement de modélisation de systèmes de production", *3ème congrès international de Génie industriel*, Tours, France, 1991.

[GOV 94] GOVINDARAJAN R., ALTMAN E. R., GAO G. R., "Minimizing register requirements under resource-constrained rate-optimal software pipelining", *27th annual International Symposium on Microarchitecture*, p. 85-94, 1994.

[GRI 72] GRINOLD R., "The Payment Scheduling Problem", *Naval Research Logistics Quarterly*, vol. 19, p. 123–136, 1972.

[GRÖ 07] GRÖFLIN H., KLINKERT A., "Feasible insertions in job shop scheduling, short cycles and stable sets", *European Journal of Operational Research*, vol. 177, num. 2, p. 763–785, 2007.

[GUÉ 00] GUÉRET C., JUSSIEN N., PRINS C., "Using intelligent backtracking to improve branch and bound methods: an application to open-shop problems", *European Journal of Operational Research*, vol. 127, num. 2, p. 344–354, 2000.

[HAA 98] HAASE K., LATTEIER J., SCHIRMER A., "The course scheduling problem at Luftansa technical training", *European Journal of Operational Research*, vol. 110, p. 441–456, 1998.

[HAG 88] HAGSTROM J., "Computational complexity of PERT problems", *Networks*, vol. 18, p. 139–147, 1988.

[HAN 95] HANEN C., MUNIER A., "Cyclic scheduling problem on parallel processors", *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, vol. 57, 1995.

[HAN 99] HANSEN P., MLADENOVIC N., "An introduction to variable neighborhood search", VOSS S., MARTELLO S., OSMAN I. H., ROUCAIROL C., Eds., *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, p. 433-458, Kluwer Academic Publishers, 1999.

[HAO 03] HAOUARI M., GHARBI A., "An improved max-flow-based lower bound for minimizing maximum lateness on identical parallel machines", *Operations Research Letters*, vol. 31, p. 49–52, 2003.

[HAR 90] HARRIS R., "Packing method for resource leveling (pack)", *Journal of Construction Engineering and Management*, vol. 116, p. 39–43, 1990.

[HAR 98a] HARTMANN S., "A competitive genetic algorithm for Resource-Constrained Project Scheduling", *Naval Research Logistics*, vol. 45, p. 733–750, 1998.

[HAR 98b] HARTMANN S., DREXL A., "Project scheduling with multiple modes: a comparison of exact algorithms", *Networks*, vol. 32, p. 283–297, 1998.

[HAR 99] HARTMANN S., *Project Scheduling Under Limited Resources - Models, Methods, and Applications*, Springer-Verlag, 1999.

[HAR 01] HARJUNKOSKI I., GROSSMANN I. E., "Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods", *Computers and Chemical Engineering*, vol. 25, p. 1647–1660, 2001.

[HAR 02] HARTMANN S., "A self-adapting genetic algorithm for project scheduling under resource constraints", *Naval Research Logistics*, vol. 49, p. 433-448, 2002.

[HER 98a] HERROELEN W., DE REYCK B., DEMEULEMEESTER E., "Resource-constrained scheduling: A survey of recent developments", *Computers and Operations Research*, vol. 25, p. 279–302, 1998.

[HER 98b] HERROELEN W., DEMEULEMEESTER E., DE REYCK B., "A classification scheme for project scheduling", WEGLARZ J., Ed., *Handbook on Recent Advances in Project Scheduling*, p. 1–26, Kluwer Academic Publishers, 1998.

[HER 99] HERROELEN W., DE REYCK B., "Phase transitions in project Scheduling", *Journal of the Operational Research Society*, vol. 150, p. 148–156, 1999.

[HER 00] HERROELEN W., DE REYCK B., E. DEMEULEMEESTER E., "On the paper 'Resource-constrained project scheduling: Notation, classification, models and methods' by Brucker et al.", *European Journal of Operational Research*, vol. 128, p. 221–230, 2000.

[HER 04] HERROELEN W., LEUS R., "Robust and reactive project scheduling: A review and classification of procedures", *International Journal of Production Research*, vol. 42, num. 8, p. 1599-1620, 2004.

[HER 05] HERROELEN W., LEUS R., "Project scheduling under uncertainty – Survey and research potentials", *European Journal of Operational Research*, vol. 165, p. 289-306, 2005.

[HER 06] HERROELEN W., "Project scheduling–theory and Practice", *Production and Operations Management*, vol. 14, num. 4, p. 413–432, 2006.

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

[ICM 94] ICMELI O., ERENGUC S., "A Tabu search procedure for Resource Constrained Project Schedule to improve project scheduling with discounted cash flows", *Computers and Operations Research*, vol. 21, p. 841–853, 1994.

[ICM 96a] ICMELI O., ERENGUC S., "A branch and bound procedure for the Resource Constrained Project Scheduling Problem with discounted cash flows", *Management Science*, vol. 42, p. 1395–1408, 1996.

[ICM 96b] ICMELI O., ERENGUC S., "The resource constrained time/cost tradeoff project scheduling problem with discounted cash flows", *Journal of Operations Management*, vol. 14, p. 255–275, 1996.

[IGE 83] IGELMUND G., RADERMACHER F., "Algorithmic approaches to preselective strategies for stochastic scheduling problem", *Network*, vol. 13, p. 29–48, 1983.

[ILO07a] ILOG SA, ILOG CP 1.2 Reference Manual, March 2007.

[ILO07b] ILOG SA, ILOG CPLEX C++ API 10.2 Reference Manual, March 2007.

[JOH 67] JOHNSON T., An algorithm for the Resource-Constrained Project Scheduling Problem, PhD thesis, M.I.T Boston (USA), 1967.

[JOH 74] JOHNSON D. S., DEMERS A. J., ULLMAN J. D., GAREY M. R., GRAHAM R. L., "Worst case performance bounds for simple one-dimensional packing algorithms", *SIAM Journal on Computing*, vol. 3, p. 299–325, 1974.

[JOH 05] JOHANNES B., "On the complexity of scheduling unit-time jobs with OR-precedence constraints", *Operations Research Letters*, vol. 33, p. 587–596, 2005.

[JOZ 98] JOZEFOWSKA J., WEGLARZ J., Eds., *Perspectives in Modern Project Scheduling*, Springer-Verlag, 1998.

[JUR 92] JURISCH B., Scheduling jobs in shops with multi-purpose machines, PhD thesis, University of Osnabrück (Germany), 1992.

[JUS 00] JUSSIEN N., DEBRUYNE R., BOIZUMAULT P., "Maintaining arc-consistency within dynamic backtracking", *Principles and Practice of Constraint Programming (CP 2000)*, num. 1894Lecture Notes in Computer Science, Singapore, Springer-Verlag, p. 249–261, 2000.

[JUS 01] JUSSIEN N., OUIS S., "User-friendly explanations for constraint programming", *ICLP'01 11th Workshop on Logic Programming Environments (WLPE'01)*, Paphos, Chypre, 2001.

[JUS 02] JUSSIEN N., LHOMME O., "Local search with constraint propagation and conflict-based heuristics", *Artificial Intelligence*, vol. 139, num. 1, p. 21–45, 2002.

[JUS 03] JUSSIEN N., The versatility of using explanations within constraint programming, Habilitation à Diriger des Recherches, Université de Nantes, Nantes (France), 2003, also available as Research Report RR-03-04 at École des Mines de Nantes.

[KAM 99] KAMMER C., *Aluminium Handbook Vol. 1: Fundamentals and Materials*, Aluminium, Düsseldorf, 1999.

[KAN 01] KANE H., BAPTISTE P., "Adjustement of the capacity to the loading rate: a dynamic model to analysis the workforce costs", *Industrial Engineering and Production Management (IEPM 2001)*, Quebec, Canada, 2001.

[KAP 88] KAPLAN L., Resource-Constrained Project Scheduling with preemption of jobs, PhD thesis, University of Michigan, Ann Arbor, MI (USA), 1988.

[KAZ 96] KAZAZ B., SEPIL C., "Project scheduling with discounted cash flows and progress payments", *Journal of the Operational Research Society*, vol. 47, p. 1262–1272, 1996.

[KEL 63] KELLEY J. E. J., "The critical-path method: resources planning and scheduling", MUTH J. F., THOMPSON G. L., Eds., *Industrial Scheduling*, p. 347-365, Prentice Hall, 1963.

[KEM 98] KEMPF K. G., UZSOY R., WANG C. S., "Scheduling a single batch processing machine with secondary resource constraints", *Journal of Manufacturing Systems*, vol. 17, p. 37–51, 1998.

[KÉR 05] KÉRI A., KIS T., "Primal-dual combined with constraint propagation for solving RCPSPWET", *Proceedings of 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'05)*, New York, U.S.A., 2005.

[KIM 01] KIMMS A., *Mathematical programming and financial objectives for scheduling projects*, Operations Research and Management Science, Kluwer Academic Publishers, 2001.

[KLE 99] KLEIN R., SCHOLL A., "Computing lower bound by destructive improvement: An application to resource-constrained project scheduling", *European Journal of Operational Research*, vol. 112, p. 322–346, 1999.

[KLE 00a] KLEIN R., "Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects", *European Journal of Operational Research*, vol. 127, p. 619–638, 2000.

[KLE 00b] KLEIN R., "Project scheduling with time-varying resource constraints", *International Journal of Production Research*, vol. 38, num. 16, p. 3937–3952, 2000.

[KLE 01] KLEIN R., *Scheduling of resource-constrained projects*, Kluwer Academic Publishers, 2001.

[KOC 03] KOCHETOV Y., STOLYAR A., "Evolutionary local search with variable neighborhood for the Resource Constrained Project Scheduling Problem", *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies (CSIT'03)*, 2003.

[KOL 91] KOLEN A., KROON L., "On the computational complexity of (maximum) class scheduling", *European Journal of Operational Research*, vol. 54, p. 23–28, 1991.

[KOL 95a] KOLISCH R., *Project scheduling under resource constraints - Efficient heuristics for several problem classes*, Physica, 1995.

[KOL 95b] KOLISCH R., SPRECHER A., DREXL A., "Characterization and generation of a general class of Resource-Constrained Project Scheduling Problems", *Management Science*, vol. 41, p. 1693–1703, 1995.

[KOL 96] KOLISCH R., "Serial and parallel Resource-Constrained Project Scheduling methods revisited: Theory and computation", *European Journal of Operational Research*, vol. 90, num. 2, p. 320-333, 1996.

[KOL 97] KOLISCH R., SPRECHER A., "PSPLIB - A project scheduling library", *European Journal of Operational Research*, vol. 96, num. 1, p. 205–216, 1997.

[KOL 99a] KOLISCH R., HARTMANN S., "Heuristic algorithms for solving the Resource-Constrained Project Scheduling Problem: classification and computational analysis", WEGLARZ J., Ed., *Handbook on Recent Advances in Project Scheduling*, Kluwer Academic Publishers, 1999.

[KOL 99b] KOLISCH R., PADMAN R., "An integrated survey of deterministic project scheduling", *Omega*, vol. 49, p. 249-272, 1999.

[KOL 99c] KOLISCH R., SCHWINDT C., SPRECHER A., "Benchmark instances for project scheduling problems", WEGLARZ J., Ed., *Handbook on Recent Advances in Project Scheduling*, p. 197-212, Kluwer Academic Publishers, 1999.

[KOL 06] KOLISCH R., HARTMANN S., "Experimental investigation of heuristics for Resource-Constrained Project Scheduling: An update", *European Journal of Operational Research*, vol. 174, num. 1, p. 23–37, 2006.

[KRÄ 95] KRÄMER A., Scheduling multiprocessor tasks on dedicated processors, PhD thesis, University of Osnabrück (Germany), 1995.

[LAB 03a] LABORIE P., "Algorithms for propagation resource constraints in AI planning and scheduling: Existing approaches and new results", *Artificial Intelligence*, vol. 143, p. 151–188, 2003.

[LAB 03b] LABORIE P., Resource temporal networks: definition and complexity, Report , ILOG, 2003.

[LAB 05] LABORIE P., "Complete MCS-Based Search: Application to Resource Constrained Project Scheduling", *Proceedings of the International Joint Conference on Artificial Intelligence, (IJCAI'05)*, Edinburgh, Scotland, 2005.

[LAM 06] LAMBRECHTS O., DEMEULEMEESTER E., HERROELEN W., Proactive and reactive strategies for the Resource-Constrained Project Scheduling Problem with uncertain resource availabilities, Report num. KBI 0606, Department of Decision Sciences & Information Management, K.U.Leuven, Belgium, 2006.

[LAW 76] LAWLER E., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[LAW 84] LAWRENCE S. R., Resource-Constrained Project Scheduling: an experimental investigation of heuristic scheduling techniques, Report , Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, USA, 1984.

[Le 94a] LE PAPE C., "Constraint-based programming for scheduling: an historical perspective", Operations Research Society seminar on Constraint Handling Techniques, 1994.

[Le 94b] LE PAPE C., "Implementation of resource constraints in ILOG Schedule: a library for the development of constraint-based scheduling systems", *Intelligent Systems Engineering*, vol. 3, num. 2, p. 55-66, 1994.

[Le 98] LE PAPE C., BAPTISTE P., "Resource constraints for preemptive job-shop scheduling", *Constraints*, vol. 3, num. 4, p. 263–287, 1998.

[LEE 96] LEE J.-K., KIM Y.-D., "Search heuristics for Resource Constrained Project Scheduling", *Journal of the Operational Research Society*, vol. 47, p. 678-689, 1996.

[LEU 03] LEUS R., The generation of stable project plans. Complexity and exact algorithms, PhD thesis, Department of applied economics, Katholieke Universiteit Leuven (Belgium), 2003.

[LEU 04] LEUS R., HERROELEN W., "Stability and resource allocation in project planning", *IIE Transactions*, vol. 36, num. 7, p. 1–16, 2004.

[LEU 05] LEUS R., HERROELEN W., "The complexity of machine scheduling for stability with a single disrupted job", *Operations Research Letters*, vol. 33, p. 151–156, 2005.

[LHO 93] LHOMME O., "Consistency Techniques for Numeric CSPs", *Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, Chambéry, France, 1993.

[LI 92a] LI K., KIM Y.-D., "An iterative scheduling technique for Resource-Constrained Project Scheduling", *European Journal of Operational Research*, vol. 56, p. 370–379, 1992.

[LI 92b] LI K., WILLIS R., "An iterative scheduling technique for Resource-Constrained Project Scheduling", *European Journal of Operational Research*, vol. 56, p. 370-379, 1992.

[LIE 08] LIESS O., MICHELON P., "A constraint programming approach for the Resource-Constrained Project Scheduling Problem", *Annals of Operations Research, to appear*, 2008.

[MAR 90] MARTELLO S., TOTH P., "Lower bound and reduction procedure for the bin-packing problem", *Discrete Applied Mathematics*, vol. 28, p. 59–70, 1990.

[MAS 70] MASTOR A., "An experimental and comparative evaluation of production line balancing techniques", *Management Science*, vol. 16, num. 11, p. 728–746, 1970.

[MCN 59] MCNAUGHTON R., "Scheduling with deadlines and loss functions", *Management Science*, vol. 6, p. 1–12, 1959.

[MEI 97] MEISELS A., GUDES E., SOLOTOREVSKY G., "Combining rules and constraints for employee timetabling", *International Journal of Intelligent Systems*, vol. 12, p. 419–439, 1997.

[MEI 03] MEISELS A., SCHAERF A., "Modeling and solving employee timetabling problems", *Annals of Mathematics and Artificial Intelligence*, vol. 39, p. 41–59, 2003.

[MER 02] MERKLE D., MIDDENDORF M., SCHMECK H., "Ant colony optimization for Resource-Constrained Project Scheduling", *IEEE Transactions on Evolutionary Computation*, vol. 6, p. 333-346, 2002.

[MIK 05] MIKA M., WALIGORA G., WEGLARZ J., "Simulated annealing and tabu search for multi-mode Resource-Constrained project Scheduling with positive discounted cash flows and different payment models", *European Journal of Operational Research*, vol. 164, p. 639–668, 2005.

[MIN 98] MINGOZZI A., MANIEZZO V., RICCIARDELLI S., BIANCO L., "An exact algorithm for the Resource-Constrained Project Scheduling Problem based on a new mathematical formulation", *Management Science*, vol. 44, num. 5, p. 714-729, 1998.

[MÖH 84] MÖHRING R., "Minimizing costs of resource requirements in project networks subject to a fixed completion time", *Operations Research*, vol. 32, p. 89–120, 1984.

[MÖH 03] MÖHRING R., SCHULZ A., STORK F., UETZ M., "Solving project scheduling problems by minimum cut computations", *Management Science*, vol. 49, num. 3, p. 330-350, 2003.

[MÖH 04] MÖHRING R. H., SKUTELLA M., STORK F., "Scheduling with AND/OR precedence constraints", *SIAM Journal on Computing*, vol. 33, p. 393–415, 2004.

[MOR 97] MORI M., TSENG C., "A genetic algorithm for multi-mode Resource-Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 100, p. 134–141, 1997.

[MOU 97] MOUKRIM A., QUILLIOT A., "A relation between multiprocessor scheduling and linear programming", *Order*, vol. 14, num. 3, p. 269–278, 1997.

[MOU 99] MOUKRIM A., "Optimal scheduling on parallel machines for a new order class", *Operations Research Letters*, vol. 24, p. 91–95, 1999.

[MOU 05] MOUKRIM A., QUILLIOT A., "Optimal preemptive scheduling on a fixed number of identical parallel machines", *Operations Research Letters*, vol. 33, p. 143–150, 2005.

[MUL 89] MULLER J., SPINRAD J., "Incremental modular decomposition", *Journal of the ACM*, vol. 36, p. 1–19, 1989.

[MUS 04] MUSLIU N., SCHAERF A., SLANY W., "Local search for shift design", *European Journal of Operational Research*, vol. 153, p. 51–64, 2004.

[NÉR 01] NÉRON E., BAPTISTE P., GUPTA J. N., "Solving hybrid flow shop problem using energetic reasoning and global operations", *Omega*, vol. 29, num. 6, p. 501–511, 2001.

[NÉR 02] NÉRON E., BAPTISTA D., "Heuristics for the multi-skill project scheduling problem", *International Symposium on Combinatorial Optimisation (CO'2002)*, Paris, France, 2002.

[NEU 99] NEUMANN K., ZIMMERMANN J., "Resource leveling for projects with schedule-dependent time windows", *European Journal of Operational Research*, vol. 117, p. 591–605, 1999.

[NEU 00a] NEUMANN K., NÜBEL H., SCHWINDT C., "Active and stable project scheduling", *Mathematical Methods of Operational Research*, vol. 52, p. 441–465, 2000.

[NEU 00b] NEUMANN K., ZIMMERMANN J., "Procedures for resource leveling and Net Present Value problems in project scheduling with general temporal and resource constraints", *European Journal of Operational Research*, vol. 127, p. 425–443, 2000.

[NEU 03] NEUMANN K., SCHWINDT C., ZIMMERMANN J., *Project Scheduling with Time Windows and Scarce Resources (2nd edition)*, Springer-Verlag, Berlin, 2003.

[NÜB 01] NÜBEL H., "The resource renting problem subject to temporal constraints", *OR Spektrum*, vol. 23, p. 359–381, 2001.

[NUI 94] NUIJTEN W., Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach, PhD thesis, Eindhoven University of Technology (The Netherlands), 1994.

[ÖZD 96] ÖZDAMAR L., ULUSOY G., "A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis", *European Journal of Operational Research*, vol. 89, p. 400-407, 1996.

[ÖZD 99] ÖZDAMAR L., "A genetic algorithm approach to a general category project scheduling problem", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 29, p. 44–69, 1999.

[PAD 97] PADMAN R., SMITH-DANIELS D., SMITH-DANIELS V., "Heuristic scheduling of Resource-Constrained Projects with cash flows", *Naval Research Logistics*, vol. 44, p. 365–381, 1997.

[PAL 04] PALPANT M., ARTIGUES C., MICHELON P., "LSSPER: solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search", *Annals of Operations Research*, vol. 131, num. 1 - 4, p. 237–257, 2004.

[PAP 79] PAPADIMITRIOU C. H., YANNAKAKIS M., "Scheduling interval-ordered tasks", *SIAM Journal on Computing*, vol. 8, num. 3, p. 405–409, 1979.

[PAS 66] PASCOE T. L., "Allocation of Resources–CPM", *Revue Française de Recherche opérationnelle*, vol. 38, p. 31–38, 1966.

[PAT 76] PATTERSON J. H., ROTH G. W., "Scheduling a project under multiple resource constraints: a zero-one programming approach", *AIIE Transactions*, vol. 8, p. 449–455, 1976.

[PAT 84] PATTERSON J. H., "A comparison of exact approaches for solving the multiple constrained Resource Project Scheduling Problem", *Management Science*, vol. 30, num. 7, p. 854–867, 1984.

[PAT 90] PATTERSON J., SLOWINSKI R., TALBOT F., WEGLARZ J., "Computational experience with a backtracking algorithm for solving a general class of precedence and resource constrained scheduling problems", *European Journal of Operational Research*, vol. 49, p. 68–79, 1990.

[PIC 95] PICOULEAU C., "New complexity results on scheduling with small communication delays", *Discrete Applied Mathematics*, vol. 60, p. 331–342, 1995.

[PIN 94] PINSON E., PRINS C., RULLIER F., "Using tabu search for solving the Resource-Constrained Project Scheduling Problem", *Proceedings of the Fourth International Workshop on Project Management and Scheduling (PMS'94)*, Leuven, Belgium, 1994.

[POD 02] PODER E., Programmation Par Contraintes et Ordonnancement de tâches avec consommation variable de ressource, PhD thesis, Université Blaise Pascal, Clermont-Ferrand (France), 2002.

[POD 04] PODER E., BELDICEANU N., SANLAVILLE E., "Computing a lower approximation of the compulsory part of a task with varying duration resource consumption", *European Journal of Operation Research*, vol. 153, p. 239-254, 2004.

[POL 06] POLICELLA N., CESTA A., ODDI A., SMITH S., "From precedence constraint posting to partial order schedules: A CSP approach to robust scheduling", AI communications, to appear, 2006.

[POT 00] POTTS C., KOVALYOV M., "Scheduling with batching: a review", *European Journal of Operational Research*, vol. 120, p. 228–249, 2000.

[PRI 69] PRITSKER A. A., WATTERS L. J., WOLFE P. M., "Multi-project scheduling with limited resources: a zero-one programming approach", *Management Science*, vol. 16, p. 93–108, 1969.

[PSPib] "Project Scheduling Problem Library", http://www.bwl.uni-kiel.de/Prod/psplib.

[QUE 94] QUEYRANNE M., SCHULZ A., Polyhedral approaches to machine scheduling, Report num. 408/1994, Technischen Universität Berlin, 1994.

[RAD 85] RADERMACHER F., "Scheduling of project networks", *Annals of Operations Research*, vol. 4, p. 227–252, 1985.

[RAU 93] RAU B., FISHER J., "Instruction-level parallel processing: history, overview, and perspective", *Journal of Supercomputing*, vol. 7, num. 1-2, p. 9–50, 1993.

[RAU 96] RAU B. R., "Iterative modulo scheduling", *International Journal of Parallel Processing*, vol. 24, num. 1, p. 3–64, 1996.

[RUS 70] RUSSELL A., "Cash flows in networks", *Management Science*, vol. 16, p. 357–373, 1970.

[RUS 86] RUSSELL R., "A comparison of heuristics for scheduling projects with cash flows and resource restrictions", *Management Science*, vol. 32, p. 1291–1300, 1986.

[SAN 99] SANKARAN J. K., BRICKER D. L., JUANG S.-H., "A strong fractionnal cutting-plane algorithm for Resource-Constrained Project Scheduling", *International Journal of Industrial Engineering: Applications and Practice*, vol. 6, num. 2, p. 99–111, 1999.

[SAU 89a] SAUER N., STONE M., "Preemptive scheduling", RIVAL I., Ed., *Algorithms and order*, p. 307-323, Kluwer Academic Publishers, 1989.

[SAU 89b] SAUER N., STONE M., "Preemptive scheduling of interval orders is polynomial", *Order*, vol. 5, num. 4, p. 345–348, 1989.

[SAV 96] SAVIN D., ALKASS S., FAZIO P., "Construction resource leveling using neural networks", *Canadian Journal of Civil Engineering*, vol. 23, p. 917–925, 1996.

[SAV 97] SAVIN D., ALKASS S., FAZIO P., "A procedure for calculating the weight-matrix of a neural network for resource leveling", *Advances in Engineering Software*, vol. 28, p. 277–283, 1997.

[SCH 80] SCHWARZE J., "An algorithm for hierarchical reduction and decomposition of a directed graph", *Computing*, vol. 25, p. 47–57, 1980.

[SCH 94] SCHIEX T., VERFAILLIE G., "Nogood recording for static and dynamic constraint satisfaction problems", *International Journal of Artificial Intelligence Tools*, vol. 3, num. 2, p. 187-207, 1994.

[SCH 00a] SCHIRMER A., "Case-based reasoning and improved adaptive search for project scheduling", *Naval Research Logistics*, vol. 47, p. 201-222, 2000.

[SCH 00b] SCHWINDT C., "Minimizing earliness-tardiness costs of resource-constrained projects", INDERFURTH K., SCHWÖDIAUER G., DOMSCHKE W., JUHNKE F., KLEINSCHMIDT P., WÄSCHER G., Eds., *Operations Research Proceedings*, Springer-Verlag, 2000.

[SCH 01] SCHWINDT C., ZIMMERMANN J., "A steepest ascent approach to maximizing the net present value of projects", *Mathematical Methods of Operations Research*, vol. 53, p. 435–450, 2001.

[SCH 03] SCHWINDT C., TRAUTMANN N., "Scheduling the production of rolling ingots: Industrial context, model, and solution method", *International Transactions in Operational Research*, vol. 10, p. 547–563, 2003.

[SCH 05] SCHWINDT C., *Resource Allocation in Project Management*, GOR-Publications, 2005.

[SEP 94] SEPIL C., "Comment on Elmaghraby and Herroelen's "The scheduling of activities to maximize the net present value of projects"", *European Journal of Operational Research*, vol. 73, p. 185–187, 1994.

[SEP 97] SEPIL C., ORTAÇ N., "Performance of the heuristic procedure for constrained projects with progress payments", *Journal of the Operational Research Society*, vol. 48, p. 1123–1130, 1997.

[SHA 65] SHAFFER L., RITTER J., MEYER W., *The Critical–Path Method*, McGraw Hill, New-York, 1965.

[SHT 97] SHTUB A., ETGAR R., "A branch and bound algorithm for scheduling projects to maximize net present value : the case of the time dependent, contingent cash flows", *International Journal of Production Research*, vol. 35, p. 3367–3378, 1997.

[SLO 89] SLOWINSKI R., WEGLARZ J., Eds., *Advances in Project Scheduling*, Elsevier, 1989.

[SMI 87] SMITH-DANIELS D., SMITH-DANIELS V., "Maximizing the net present value of a project subject to materials and capital constraints", *Journal of Operations Management*, vol. 7, p. 33–45, 1987.

[SMI 94] SMITH S. F., "Reactive scheduling systems", BROWN D., SCHERER W., Eds., *Intelligent Scheduling Systems*, Kluwer Academic Publishers, 1994.

[SMI 96] SMITH-DANIELS D., PADMAN R., SMITH-DANIELS V., "Heuristic scheduling of capital constrained projects", *Journal of Operations Management*, vol. 14, p. 241–254, 1996.

[SPR 94] SPRECHER A., "Resource-Constrained Project Scheduling: exact methods for the multi-mode case", *Lecture Notes in Economics and Mathematical Systems, 409*, vol. 409, 1994.

[SPR 95] SPRECHER A., KOLISCH R., DREXL A., "Semi-active, active, and non-delay schedules for the Resource-Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 80, num. 1, p. 94-102, 1995.

[SPR 00] SPRECHER A., "Scheduling resource-constrained projects competitively at modest memory requirements", *Management Science*, vol. 46, p. 710-123, 2000.

[STI 78] STINSON J. P., DAVIS E. W., KHUMAWALA B. M., "Multiple resource-constrained scheduling using branch-and-bound", *AIIE Transactions*, vol. 10, num. 3, p. 252–259, 1978.

[STO 05] STORK F., UETZ M., "On the generation of circuits and minimal forbidden sets", *Mathematical Programming*, vol. 102, num. 1, p. 185-203, 2005.

[TAL 78] TALBOT F., PATTERSON J. H., "An efficient integer programming algorithm with network cuts for solving RCSP", *Management Science*, vol. 24, num. 11, p. 1163–1174, 1978.

[TAL 82] TALBOT F., "Resource-Constrained Project Scheduling with time-resource tradeoffs : the non preemptive case", *Management Science*, vol. 28, p. 1197–1210, 1982.

[TER 04] TERCINET F., LENTÉ C., NÉRON E., "Mixed satisfiability tests for multiprocessor scheduling with release dates and deadlines", *Operations Research Letters*, vol. 32, num. 11, p. 326–330, 2004.

[THO 98] THOMAS P., SALHI S., "A tabu search approach for the Resource Constrained Project Scheduling Problem", *Journal of Heuristics*, vol. 4, p. 123-139, 1998.

[TOR 03] TOROSLU I., "Personnel assignment problem with hierarchical ordering constraints", *Computers and Industrial Engineering*, vol. 45, p. 493–510, 2003.

[TSA 93] TSANG E., *Foundations of constraint satisfaction*, Academic Press, 1993.

[UET 01] UETZ M., Algorithms for Deterministic and Stochastic Scheduling, PhD thesis, Technische Universität Berlin (Germany), 2001.

[ULU 00] ULUSOY G., CEBELLI S., "An equitable approach to the payment scheduling problem in project management", *European Journal of Operational Research*, vol. 127, p. 262–278, 2000.

[UZS 95] UZSOY R., "Scheduling batch processing machines with incompatible job families", *International Journal of Production Research*, vol. 33, p. 2685–2708, 1995.

[VAL 03] VALLS V., QUINTANILLA M., BALLESTIN F., "Resource-Constrained Project Scheduling: A critical reordering heuristic", *European Journal of Operational Research*, vol. 149, p. 282-301, 2003.

[VAL 04] VALLS V., QUINTANILLA M., BALLESTIN F., "A population-based approach to the Resource-Constrained Project Scheduling Problem", *Annals of Operations Research*, vol. 131, p. 305-324, 2004.

[VAL 05] VALLS V., BALLESTIN F., QUINTANILLA M., "Justification and RCPSP: A technique that pays", *European Journal of Operational Research*, vol. 165, p. 375-386, 2005.

[VAN 99] VAN HENTENRYCK P., *The OPL Optimization Programming Language*, MIT Press, 1999.

[VAN 01a] VANHOUCKE M., DEMEULEMEESTER E., HERROELEN W., "An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem", *Annals of Operations Research*, vol. 102, p. 179–196, 2001.

[VAN 01b] VANHOUCKE M., DEMEULEMEESTER E., HERROELEN W., "Maximizing the net present value of a project with linear time-dependent cash flows", *International Journal of Production Research*, vol. 39, p. 3159–3181, 2001.

[VAN 01c] VANHOUCKE M., DEMEULEMEESTER E., HERROELEN W., "On maximizing the net present value of a project under renewable resource constraints", *Management Science*, vol. 47, p. 1113–1121, 2001.

[VAN 03] VANHOUCKE M., DEMEULEMEESTER E., HERROELEN W., "Progress payments in project scheduling problems", *European Journal of Operational Research*, vol. 148, p. 604–620, 2003.

[VAN 04] VANHOUCKE M., COELHO J., DEBELS D., TAVARES L. V., On the morphological structure of a network, Vlerick Leuven Gent Working Paper Series num. 2004/9, Vlerick Leuven Gent Management school, 2004.

[Van 05] VAN DE VONDER S., DEMEULEMEESTER E., HERROELEN W., Heuristic procedures for generating stable project baseline schedules, Report num. 0516, Department of Applied Economics, K.U.Leuven, Belgium, 2005.

[Van 06a] VAN DE VONDER S., BALLESTÍN F., DEMEULEMEESTER E., HERROELEN W., Heuristic procedures for reactive project scheduling, Report num. KBI 0605, Department of Decision Sciences & Information Management, K.U.Leuven, Belgium, 2006.

[Van 06b] VAN DE VONDER S., DEMEULEMEESTER E., HERROELEN W., LEUS R., "The trade-off between stability and makespan in Resource-Constrained Project Scheduling", *International Journal of Production Research*, vol. 44, num. 2, p. 215-236, 2006.

[Van 06c] VAN DE VONDER S., DEMEULEMEESTER E., LEUS R., HERROELEN W., "Proactive-reactive project scheduling - Trade-offs and procedures", WEGLARZ J., JOZEFOWSKA J., Eds., *Perspectives in Modern Scheduling*, vol. 92 of *International Series in Operations Research & Management Science*, Springer-Verlag, 2006.

[VAN 06d] VANHOUCKE M., "An efficient hybrid search procedure for various optimization problems", *Lecture Notes in Computer Science*, vol. 3906, p. 272–283, 2006.

[Van 08a] VAN DE VONDER S., DEMEULEMEESTER E., HERROELEN W., "An investigation of efficient and effective predictive-reactive project scheduling procedures", *Journal of Scheduling, to appear*, 2008.

[VAN 08b] VANHOUCKE M., DEBELS D., "The discrete time/cost trade-off problem: extensions and heuristic procedures", *Journal of Scheduling, to appear*, 2008.

[VER 94] VERFAILLIE G., SCHIEX T., "Solution reuse in dynamic constraint satisfaction problems", *Proc. of the 12th National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA, USA, p. 307-312, 1994.

[VER 05] VERFAILLIE G., JUSSIEN N., "Constraint solving in uncertain and dynamic environments – a survey", *Constraints*, vol. 10, num. 3, p. 253–281, 2005.

[VIE 00] VIEIRA G., HERRMANN J., LIN E., "Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies", *International Journal of Production Research*, vol. 38, num. 8, p. 1899–1915, 2000.

[VIE 03] VIEIRA G., HERRMANN J., LIN E., "Rescheduling manufacturing systems: A framework of strategies, policies, and methods", *Journal of Scheduling*, vol. 6, num. 1, p. 35–58, 2003.

[VIL 86] VILAIN M., KAUTZ H., "Constraint propagation algorithms for temporal reasoning", *Proceedings of Fifth National Conference on Artificial Intelligence*, p. 377–382, 1986.

[VIL 04] VILIM P., "O(n log n) Filtering Algorithms for Unary Resource Constraint", *Proceedings of the 3th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR'04*, p. 319-334, 2004.

[WAN 05] WANG J., "Constraint-based schedule repair for product development projects with time-limited constraints", *International Journal of Production Economics*, vol. 95, p. 399-414, 2005.

[WEG 98] WEGLARZ J., Ed., *Project Scheduling - Recent Models, Algorithms and Applications*, Springer-Verlag, 1998.

[WU 93] WU S. D., STORER R. H., CHANG P. C., "One-machine rescheduling heuristics with efficiency and stability as criteria", *Computers and Operations Research*, vol. 20, p. 1–14, 1993.

[YAN 92] YANG K., TALBOT F., PATTERSON J., "Scheduling a project to maximize its net present value: an integer programming approach", *European Journal of Operational Research*, vol. 64, p. 188–198, 1992.

[YAN 95] YANG K., TAY L., SUM C., "A comparison of stochastic scheduling rules for maximizing project net present value", *European Journal of Operational Research*, vol. 85, p. 327–339, 1995.

[YOU 96] YOUNIS M., SAAD B., "Optimal resource leveling of multi-resource projects", *Computers and Industrial Engineering*, vol. 31, p. 1–4, 1996.

[YU 04] YU G., QI X., *Disruption management - framework, models and applications*, World Scientific, New Jersey, 2004.

[ZHU 97] ZHU D., PADMAN R., "Connectionist approaches for solver selection in constrained project scheduling", *Annals of Operations Research*, vol. 72, p. 265–298, 1997.

[ZHU 05] ZHU G., BARD J., YU G., "Disruption management for Resource-Constrained Project Scheduling", *Journal of the Operational Research Society*, vol. 56, p. 365-381, 2005.

Contributing authors

Christian Artigues, Université de Toulouse, LAAS-CNRS,
7 avenue du Colonel Roche 31077 Toulouse Cedex 4, France.

Sadia Azem École des Mines de Saint-Etienne[1] and Université du Luxembourg[2],
[1] *École des Mines de Saint-Etienne, CMP Georges Charpak, Avenue des Anémones - Quartier Saint-Pierre, F-13541 Gardanne- France*
[2]*Université du Luxembourg, Faculté des Sciences, de la Technologie et de la Communication, 6, rue Richard Coudenhove-Kalergi, L-1511 Luxembourg*

Odile Bellenguez-Morineau Université François Rabelais Tours[1] and Ecole des Mines de Nantes, IRCCyN[2]
[1] *Laboratoire d'Informatique, Polytech'Tours, 64 avenue Jean Portalis F-37200, Tours, France*
[2] *Ecole des Mines de Nantes - Dep. d'Automatique et Productique, 4 rue Alfred Kastler, La Chantrerie BP 20722 44307 Nantes Cedex 3 (present address).*

Jean-Charles Billaut, Université François Rabelais Tours,
Laboratoire d'Informatique, Polytech'Tours, 64 avenue Jean Portalis, 37200 Tours, France.

Cyril Briand, Université de Toulouse, LAAS-CNRS,
7 avenue du Colonel Roche 31077 Toulouse Cedex 4, France.

Jacques Carlier, Université de Technologie de Compiègne,
HeuDiaSyC, UMR CNRS 6599, Génie Informatique, BP 20529, 60205 Compiègne cedex France.

Jean Damay, Blaise Pascal University, LIMOS[1] and CIRRELT[2]
[1]*LIMOS, Complexe scientifique des Cézeaux, 63177 Aubière CEDEX, France,*

[2]*CIRRELT, Université de Montréal C.P. 6128, succ. Centre-ville Montréal, QC H3C 3J7 Canada (present address).*

Sophie Demassey, École des Mines de Nantes,
4 rue Alfred Kastler, BP 20722, F-44307 Nantes Cedex 03, France.

Erik Demeulemeester, Katholieke Universiteit Leuven,
Faculty of Business and Economics, Department of Decision Sciences and Information Management, Research Center for Operations Management, Naamsestraat 69, BE-3000 Leuven, Belgium.

Laure-Emmanuelle Drezet, Université François Rabelais Tours,
Laboratoire d'Informatique, Polytech'Tours, 64 avenue Jean Portalis, 37200 Tours, France.

Benoît Dupont de Dinechin, STMicroelectronics STS/CEC
12, rue Jules Horowitz - BP 217. F-38019 Grenoble

Michel Gourgand, Blaise Pascal University, LIMOS,
ISIMA, Campus des Cézeaux, 63173 Aubière Cedex, France.

Nathalie Grangeon, Blaise Pascal University, LIMOS
IUT de Montluçon, Avenue Aristide Briand, 03100 Montluçon, France.

Christelle Guéret, École des Mines de Nantes,
4 rue Alfred Kastler, BP 20722, F-44307 Nantes Cedex 03, France.

Willy Herroelen, Katholieke Universiteit Leuven,
Faculty of Business and Economics, Department of Decision Sciences and Information Management, Research Center for Operations Management, Naamsestraat 69, 3000 Leuven, Belgium.

Narendra Jussien, École des Mines de Nantes, LINA-CNRS
4 rue Alfred Kastler, BP 20722, F-44307 Nantes Cedex 03, France.

Oumar Koné, Université de Toulouse, LAAS-CNRS,
7 avenue du Colonel Roche 31077 Toulouse Cedex 4, France.

Roel Leus, Katholieke Universiteit Leuven,
Faculty of Business and Economics, Naamsestraat 69, 3000 Leuven, Belgium.

Pierre Lopez, Université de Toulouse, LAAS-CNRS,
7 avenue du Colonel Roche 31077 Toulouse Cedex 4, France.

Marcel Mongeau, Université de Toulouse, UPS, Institut de Mathématiques
31062 Toulouse cedex 9 France

Aziz Moukrim, Université de Technologie de Compiègne,
HeuDiaSyC, UMR CNRS 6599, Génie Informatique, BP 20529, 60205 Compiègne cedex France.

Emmanuel Néron, Université François Rabelais Tours,
Laboratoire d'Informatique, Polytech'Tours, 64 avenue Jean Portalis F-37200, Tours, France.

Sylvie Norre, Blaise Pascal University, LIMOS,
IUT de Montluçon, Avenue Aristide Briand, 03100 Montluçon, France.

Wim Nuijten, ILOG,
9 rue de Verdun, 94253 Gentilly Cedex France.

David Rivreau, Institut de Mathématiques Appliquées - Université Catholique de l'Ouest
3 place André-Leroy BP 10808 - 49008 Angers Cedex 01, France.

Christoph Schwindt, Clausthal University of Technology, Institute of Management and Economics,
Julius-Albert-Str. 2, 38678 Clausthal-Zellerfeld, Germany.

Norbert Trautmann, University of Bern,
Departement Betriebswirtschaftslehre, Schuetzenmattstrasse 14, 3012 Bern, Switzerland.

Huang Xu, Université de Technologie de Compiègne,
HeuDiaSyC, UMR CNRS 6599, Génie Informatique, BP 20529, 60205 Compiègne cedex France.

Index