Université de Tunis
Institut Supérieur de Gestion

Thèse présentée en vue de l'obtention du titre de

Docteur en Gestion

Exact Algorithms for Scheduling Parallel Machines with Heads and Tails

Elaborée par :

Anis Gharbi

Soutenue le 27 février 2003

Devant le jury composé de :

Pr. Khaled Mellouli, Président
Pr. Mohamed Haouari, Directeur de Thèse
Pr. Foued Ben Abdelaziz, Rapporteur
Pr. Khaled Ghedira, Rapporteur
Pr. Habib Chabchoub, Examinateur

to my mother,

to my wife,

to my son

Acknowledgements

First, I would like to express my profound gratitude to my supervisor, Professor Mohamed Haouari. His ever growing proficiency, tight supervision, and very high human qualities, let me take an immense pleasure in doing research and incited me to give my best throughout the preparation of this thesis.

I would never have discovered this extremely interesting field, which is operations research, without the valuable contribution of all my professors of the High Institute of Management (ISG) of Tunis to whom I sincerely owe many thanks. Also, I would like to acknowledge the kind collaboration of the staff of the Polytechnical School of Tunisia.

Special thanks are affectionately aimed to my colleagues Talel Ladhari, Mohamed Jeddy, and Jouhaina Chaouachi Siala for their friendship and their help.

I express my boundless acknowledgement to my adorable wife Sana. Her care, patience and encouragements were really a precious support. Needless to say that such a happy event would never have happened without the irreplaceable presence of my tender mother, my kind father, and my affectionate brothers and sisters, to whom I am eternally grateful.

Anis Gharbi

Contents

Chapter 1

Introduction

1.1 Scheduling theory

In most manufacturing systems, a decision-making process that plays a crucial role consists in allocating the time at which a particular task is to be processed by a given resource in order to optimize the requirements set by the customer. This function is referred to as *scheduling.* Although a manufacturing system incorporates several other key functions (such as finance, human resources management, capacity planning, materials management, quality control, production process design, maintenance, etc...), the importance of scheduling has increased in recent years. Indeed, the current economic and commercial market pressures (the growing consumer demand for variety, reduced product life cycle, changing markets with global competition, rapid development of new processes and technologies, etc...) emphasize the need for a system which requires only small inventory levels, minimizes waste production but is able to maintain customer satisfaction by delivering the required goods at the specified time. This requires efficient, effective and accurate scheduling, which is a complex operation in almost all production environments. The importance of scheduling is exemplified by an investigation carried out in the United States mechanical industrial sector which shows that the machines spend about 80% of their total processing time in waiting for the tasks.

Needless to say that the maintenance of a profitable, vigorous and buoyant manufacturing sector is essential for the prosperity of a country. Nevertheless, the importance of the scheduling function is not solely confined to the manufacturing activity. Indeed, scheduling problems also exist in information-processing environments as well as several service industries such as in transportation and distribution settings.

Scheduling theory originated as a research area in the early 1950s. Some of the first scheduling publications which appeared in operations research literature are the pioneering results of Johnson (1954), Jackson (1956), and Smith (1956). During the 1960s, a significant amount of work had been performed on dynamic programming and integer programming formulations of scheduling problems. After Karp's famous paper on complexity theory (1972), the research in the 1970s focused mainly on complexity hierarchy of scheduling problems. In the 1980s, several different directions were pursued in academy and industry. Since then, the field has attracted a lot of attention and an impressive amount of literature has been created. Indeed, besides the increasing papers dealing with scheduling and published in several operations research journals (such as European Journal of Operational Research, Operations Research, Discrete Applied Mathematics, Operations Research Letters, Annals of Operations Research, to quote just a few), two specialized international journals have been recently created: *Journal of Scheduling* and *IIE Transactions on Scheduling and Logistics*.

Scheduling area has become an important branch of combinatorial optimization, located at the interface between applied mathematics, computer science, and operations research. The surprising difficulty of scheduling problems continues to intrigue researchers which increases their theoretical interest. Besides, the area of scheduling is motivated by practical considerations and has so led to successful theoretical investigations. Scheduling area will undoubtedly continue to provide opportunities for fruitful interaction between theory and practice.

Most research has been traditionally focused on *deterministic* scheduling which already allows for more than enough variety. In contrast to *stochastic* scheduling problems, deterministic problems are based on the restrictive assumption that all task parameters are known with certainty in advance. Although processing times and task arrivals may be subject to fluctuations, the deterministic assumption may be suitable for several practical situations. Indeed, in many cases, these fluctuations are of no significant impact on the quality of the schedule. Also, the deterministic assumption is often imposed by certain applications such as in computer control systems working in a hard-real-time environment (Bertossi and Fusiello, 1997). Moreover, there are (surprisingly) many cases where a simple rule which is merely a heuristic for the deterministic model has a stochastic reformulation which solves the stochastic model to optimality (see for instance Lawler et al. 1993). In this thesis, we focus only on deterministic scheduling.

For a comprehensive survey of scheduling theory, the reader is referred to Conway et al. (1967), Baker (1974), Nabeshima et al. (1982), Blazewicz (1987), Blazewicz et al. (1988), Blazewicz et al. (1991), Lawler et al. (1993), Morton and Pentico (1993), Blazewicz et al. (1994b), Tanaev et al. (1994a, 1994b), Parker (1995), Pinedo (1995), Brucker (1998), and Pinedo and Chao (1998).

Also, several papers deal with real-life scheduling applications such as Rosenbloom and Goertzen (1987), Buxey (1989), Dodin and Chan (1991), Jarrah et al. (1992), Alfa et al. (1994), Wiers (1997), Rochat (1998), Basnet et al. (1999), Della Croce et al. (1999), Hart et al. (1999), Brucker and Hurink (2000), Pendharkar and Rodger (2000), Ferland et al. (2001), and Xu and Chiu (2001), to quote just a few.

In order to position the problem dealt with in this thesis, and before its formal description, we provide a brief overview of scheduling models.

1.2 Scheduling models

Scheduling theory is generally concerned with the optimal allocation of scarce resources to activities over time. More formally, scheduling problems involve *jobs* that must be scheduled on *machines* subject to certain constraints to optimize some objective function. The goal is to specify a schedule that specifies when and on which machine each job is to be executed.

As scheduling problems arise in a variety of settings, job and machine entities can easily be replaced with doctors and patients, students and classrooms, airline crews and aeroplanes, docks and ships, etc...The following examples illustrate the role of scheduling in two different real-life situations.

Example 1.1: Consider the central processing unit of a computer that must process a sequence of programs (jobs) that arrive over time. In what ordering should the programs be processed in order to minimize the average completion time?

Example 1.2: Consider a factory that produces paper bags for cement, charcoal, dog food, and so on. The basic raw material for such an operation is rolls of papers. The production process consists of three stages: printing the logo, gluing the side of the bag, and sewing up one end or both ends. The different bags require different amounts of processing times on different machines. The factory has orders for batches of bags; each order has a date

by which it must be completed. In what ordering should the machines work on different bags in order to ensure that the factory completes as many orders as possible on time?

Graham et al. (1979) introduced the standard $\alpha|\beta|\gamma$ notation for representing scheduling problems. This notation embodies the three main elements which define the scheduling problem: the machine environment, the job characteristics, and the optimization criterion. In the sequel, we briefly detail these three fields. Clearly, the scheduling domain is very large, encompassing a diverse range of models, so any discussion of the available material has to be selective (for a broader description of scheduling models, the reader is referred to Graham et al. (1979)). In the considered scheduling models, the number of machines and the number of jobs are assumed to be finite and fixed.

1.2.1 Machine environments

There are several machine environments (represented by the field α) which are summarized in the following :

- *Single machine* ($\alpha = 1$)**:** It is the simplest special case of all possible machine environments
- *Parallel machines***:** Each job requires a single operation to be performed on one out of a set of available machines. This environment can be divided into three classes:
  - *Identical parallel machines* ($\alpha = P$)*:* All the available machines have the same speed
  - *Uniform parallel machines* ($\alpha = Q$)*:* The machines have different speeds, but these speeds are independent of the jobs
  - *Unrelated parallel machines* ($\alpha = R$)*:* The machines have different speeds, but these speeds are dependent of the jobs
- *Flow shop* ($\alpha = F$)**:** There are several machines in series. Each job has to be processed on each one of the machines. All jobs have the same routing (that is, they have to be processed first on machine 1, then on machine 2, and so on)
- *Job shop* ($\alpha = J$): This model is similar to the flow shop, with the only difference that each job has its own route to follow

- *Open shop* ($\alpha = O$): Likewise the job shop, each job has to be processed on each one of the machines. However, there is no restriction on the routing of each job. The scheduler is allowed to determine the route of any job

Some additional information on the machine environment may be pointed out in the field α. For instance, $\alpha = F2$ specifies that there are two machines in the considered flow shop. Another example is when the identical parallel machines are not continuously available and the number of available machines increases with time. In this case, we set $\alpha = P, NC_{inc}$.

1.2.2 Job characteristics

Several possible job characteristics (represented by the field β) may modify the scheduling environment. Some of these characteristics are:

- *Preemption* (*pmtn*): The processing of any operation may be interrupted and resumed at a later time
- *Precedence constraints* (*prec*): A precedence relation between jobs requires that one or more jobs have to be completed before another job is allowed to start its processing
- *Release dates or heads* (r_j): No job can start its processing before its release date
- *Delivery times or tails* (q_j): After finishing its processing, each job has to spend an amount of time before exiting the system

1.2.3 Optimization criteria

The goal of a scheduling algorithm is to produce a "good" schedule, but the definition of "good" will vary depending on the application. Therefore, an optimization criterion (represented by the field γ) has to be specified. The most commonly chosen criteria involve the minimization of:

- *Makespan* ($C_{\max}$): The completion time of the last job to leave the system
- *Maximum lateness* ($L_{\max}$): The worst violation of the due dates. The job lateness is non-negative if it is completed late and negative otherwise

- *Maximum tardiness* ($T_{\max}$): The difference between tardiness and lateness is that tardiness equals zero if the job is completed early (i.e. $T_{\max} = \max(0, L_{\max})$)

- *Maximum flow time* ($F_{\max}$): The flow time of a job denotes the time elapsed between its entry to its exit from the system

- *Total (weighted) completion time* ($\sum C_j$ or $\sum w_j C_j$)**:** The sum of the (weighted) completion times. It indicates the total holding (or inventory) costs incurred by the schedule. This criterion is equivalent to the total (weighted) flow time criterion

- *Total (weighted) tardiness* ($\sum T_j$ or $\sum w_j T_j$): It is a more general cost function than the total (weighted) completion time

- *(Weighted) Number of tardy jobs* ($\sum U_j$ or $\sum w_j U_j$): A job is considered as tardy if it is completed after its due date

It is worth noting that some job characteristics may be implicitly indicated by the optimization criterion. For instance, the lateness or the number of tardy jobs criterion involves that each job is preferred to be completed before its due date. Also, the total weighted completion time criterion means that each job has a priority factor which denotes its importance relatively to the other jobs.

For instance, the problems described in examples 1.1, and 1.2 are denoted by $1||\sum C_j$ and $F3||\sum U_j$, respectively.

1.3 Problem description and motivations

The attention of this thesis is focused on the model of parallel machine scheduling. In particular, we are interested in the development, design and analysis of new exact algorithms for the following scheduling problem:

We are given a set J of n jobs that have to be scheduled on a set of m identical parallel machines ($n > m \geq 2$) under the following constraints:

- The processing of job j cannot be started before its release date (or *head*) r_j, which denotes the time at which the job j enters the system

- Each job j has to be processed for p_j units of time by one machine out of the set of machines

- Each job j has a delivery time (or *tail)* q_j that must elapse between its completion on the machine and its exit from the system
- All data are assumed to be integer and deterministic
- All machines are continuously available from time zero onwards
- Each machine processes at most one job at one time
- Each job cannot be processed by more than one machine at one time
- Preemption is not allowed

The completion time of a schedule (or *makespan*) is defined as

$$C_{\max} = \underset{j=1,\ldots,n}{Max} \ (t_j + p_j + q_j)$$

where t_j denotes the start time of job j. The objective is to find a feasible schedule of minimum completion time. Using the notation of Graham et al. (1979), this problem is denoted by $P|r_j, q_j|C_{\max}$.

The $P|r_j, q_j|C_{\max}$ provides a useful problem for analysis because it is considered as a good representation of the parallel machine scheduling domain. Indeed, it constitutes a fundamental model encompassing several well studied parallel machine scheduling problems, including $P||C_{\max}$, $P|r_j|C_{\max}$, $P||L_{\max}$, and the parallel machine problem with machine availabilities denoted by $P, NC_{inc}||C_{\max}$. Hence, many of the results presented in this thesis might be applicable for all these problems. Also, the $P|r_j, q_j|C_{\max}$ has many interesting theoretical applications. In particular, it arises as a strong relaxation of the Multiprocessor Flow Shop problem (Vandevelde 1994, Hoogeveen et al. 1995, Perregaard 1995) and it plays a central role in some exact algorithms for the Resource Constrained Project Scheduling Problem (Carlier and Latapie 1991). Thus, the challenge in developing an efficient exact algorithm for the $P|r_j, q_j|C_{\max}$ is to achieve a significant step toward the exact solution of other scheduling problems. It is worth noting that the only existing exact algorithm for the $P|r_j, q_j|C_{\max}$ is the branch-and-bound algorithm proposed by Carlier (1987).

Besides the important aforementioned theoretical applications, the $P|r_j, q_j|C_{\max}$ has also several interesting practical applications. In particular, many production environments contain a single bottleneck stage which cannot process the good or service quickly enough to prevent backlogs. In this case, an efficient scheduling of the system requires as a first step the optimal

scheduling of the bottleneck center, and then proceeding with the scheduling of the process located before and after the bottleneck. This approach is referred to as the *theory of constraints* (Goldratt and Cox, 1992). The OPT (*Optimized Production Technology*) software developed by Eliayahu Goldratt in the late 1970s, is considered the genesis of bottleneck systems (Sipper and Buflin, 1998, p.531). A number of software packages to schedule bottleneck systems are available including OPT21, OPIS, MICRO-BOSS, MOOPI, and PRIORITY (Davis et al. 1999). Clearly, an optimal scheduling of the bottleneck requires the optimization of a $P|r_j, q_j|C_{\max}$. Therefore, improvements of the optimization methodology of the $P|r_j, q_j|C_{\max}$ may result in the improvement of the efficiency of scheduling bottleneck systems.

1.4 Organization of the thesis

The thesis is organized as follows. In Chapter 2, we draw up an exhaustive survey of the parallel machine scheduling problems. The basic complexity results as well as the most important (exact and approximate) optimization algorithms proposed for these problems are reported. Chapter 3 is devoted to the presentation of some mathematical properties of the $P|r_j, q_j|C_{\max}$ as well as the design of a preprocessing algorithm devised to simplify the problem. In Chapter 4, we investigate lower bounding strategies for the $P|r_j, q_j|C_{\max}$. These bounds will be used to design efficient branch-and-bound algorithms. A first exact algorithm for the $P|r_j, q_j|C_{\max}$ is introduced in Chapter 5. Extensive computational experiments show that this algorithm consistently outperforms Carlier's one. In Chapter 6, a second branch-and-bound algorithm based on a new formulation of the problem is detailed. Our experimental results show that this new algorithm makes it feasible to solve instances with up to 1500 jobs and 50 machines in a small CPU time. Finally, we conclude our report by providing a synthesis of our research and indicating some directions for future research.

Chapter 2

A survey of parallel machine scheduling problems

In this chapter, we present a brief overview of the research devoted to the parallel machine scheduling problems with emphasis on the makespan criterion. For a more detailed discussion regarding this ever growing field, the reader is referred to Cheng and Sin (1990) and Mokotoff (2001).

2.1 Complexity results

Most of scheduling problems belong to the wide class of Combinatorial Optimization problems known to be intractable (in the sense that it is not likely that there exist efficient optimization algorithms to solve them). More formally, these problems are said to be $\mathcal{NP}$-hard (Garey and Johnson, 1979; Papadimitriou, 1994). Only few scheduling problems have been shown to be "easily solvable". What means that they are solvable in polynomial time. In this section, we briefly review some complexity results related to the parallel machine scheduling problems.

A very fundamental problem in scheduling theory is the classical identical parallel machine problem denoted by $P||C_{\max}$. This problem is $\mathcal{NP}$-hard for the case of two machines (Karp, 1972), and strongly $\mathcal{NP}$-hard for larger number of machines (Garey and Johnson, 1978). However, the preemptive version $P|pmtn|C_{\max}$ is solvable in linear time (McNaughton, 1959). The preemptive problem remains easy even in the presence of release dates. Indeed, Gonzalez and Johnson (1980) presented an $O(mn)$ algorithm which provides an optimal solution of the $P|r_j, pmtn|C_{\max}$.

$P|r_j, q_j|C_{\max}$ is an extension of the classical $P||C_{\max}$. A formal proof of the $\mathcal{NP}$-hardness of $P|r_j, q_j|C_{\max}$ is provided in Chapter 3. However, the preemptive version denoted by $P|r_j, q_j, pmtn|C_{\max}$ is solvable in polynomial time using a network flow formulation (Horn, 1974) (see chapter 4 on page 46). Another interesting polynomially-solvable version is the case where all jobs have equal processing time, which is solvable in $O(mn^2)$ time (Simons and Warmuth, 1989).

It is worth noting that an $\mathcal{NP}$-hard identical parallel machine problem remains $\mathcal{NP}$-hard if the machines have different speeds. Now, we briefly review the polynomial solvable cases of uniform and unrelated machines. Dessouky et al. (1990) proved that the unit length uniform machine problem is solvable in $O(n \log n)$ time even in the presence of release dates or delivery times. The preemptive uniform machine problem denoted by $Q|pmtn|C_{\max}$ is solvable in linear time (Gonzalez and Sahni, 1978). If release dates or delivery times are added to the latter problem, Labetoulle et al. (1984) proved that it is solvable in $O(n \log n + nm)$ time. The problem becomes slightly more difficult if both release dates and delivery times are considered. Federgruen and Groenevelt (1986) used a maximum-flow formulation to derive an $O(tn^3(\log n + \log s_{\max} + \log p_{\max}))$ algorithm for solving the $Q|pmtn, r_j, q_j|C_{\max}$, where t denotes the number of distinct speeds, and $s_{\max}$ and $p_{\max}$ denote the largest machine speed and processing time, respectively. However, the two-machine case can be solved in $O(n^3)$ using the algorithm proposed by Labetoulle et al. (1984).

Lawler and Labetoulle (1978) used a linear programming formulation to prove that the preemptive version of the unrelated parallel machine problem is polynomially solvable. De Werra (1990) showed that under some conditions, this approach can also solve the nonpreemptive case. The particular case of two machines, denoted by $R2|pmtn|C_{\max}$, has been shown to be solvable in linear time (Gonzalez et al., 1990).

2.2 Exact algorithms

Several exact approaches have been proposed for the $P||C_{\max}$. In particular, an efficient branch-and-bound algorithm was proposed by Dell'Amico and Martello (1995). Ho and Wong (1995) proposed an interesting exact algorithm for the two-machine case. Also, Mokotoff (2002) proposed a promising polyhedral approach. It is worth noting that several dynamic programming schemes have been proposed in the literature but can only solve tiny instances (Rothkopf, 1966; Sahni, 1976).

On the other hand, despite its theoretical and practical interest, the literature regarding the $P|r_j, q_j|C_{\max}$ is relatively scarce. Indeed, the only exact algorithm proposed in the literature for this problem is that developed by Carlier (1987). This algorithm is fully discussed in Chapter 5 (see page 91). However, several improved branch-and-bound algorithms have been proposed for the single machine problem $1|r_j, q_j|C_{\max}$. In particular, the algorithm proposed by Carlier (1982) is considered very efficient. Bratley et al. (1975) developed a simple enumerative algorithm for the problem $P|r_j, \bar{d}_j|C_{\max}$ where $\bar{d}_j$ represents the deadline of job j (i.e. the job *must* be completed before this date). Their algorithm was able to solve only small instances.

The uniform parallel machine version was investigated by Dessouky (1998) who developed a branch-and-bound algorithm for $Q|r_j, q_j, p_j = p|C_{\max}$. He reported the solution of instances with up to 80 jobs and 3 machines. One of the first exact algorithms for the unrelated parallel machines problem $R||C_{\max}$ is the dynamic programming algorithm proposed by Horowitz and Sahni (1976). Van de Velde (1993) presented an optimization algorithm based on surrogate relaxation and duality. Martello et al. (1997) proposed a branch-and-bound algorithm for the $R||C_{\max}$. Recently, Mokotoff and Chrétienne (2002) presented a very efficient exact algorithm using polyhedral approach. The problem with two unrelated machines, denoted by $R2|r_j, q_j|C_{\max}$, was addressed by Lancia (2000) who gave an enumerative method.

2.3 Heuristics

2.3.1 Constructive algorithms

A constructive algorithm builds a solution, starting from the input data, following a set of rules. The basic constructive algorithms are the so-called *List Schedules* (Graham, 1966): A list with all jobs is made using some dispatching rule, and the uppermost job is assigned to the first available machine until all the jobs are scheduled. Graham (1966) proved that, for the $P||C_{\max}$, the ratio of the makespan provided by any list schedule over the optimal makespan is not larger than $2 - \frac{1}{m}$. This smart result can be seen as follows:

Let LS be any list schedule and denote by $C_{\max}(LS)$ its makespan. Let j_0 denote the last job to be completed in LS. Let $J_i \subset J \backslash \{j_0\}$ denote the set of jobs processed by machine i ($i = 1, ..., m$) in LS. Note that no machine can be idle before the starting time t_0 of j_0. That is, $\sum_{j \in J_i} p_j \geq t_0$ for $i = 1, ..., m$.

Therefore, $\sum_{j \neq j_0} p_j \geq m t_0$. Consequently,

$$C_{\max}(LS) = t_0 + p_{j_0} \leq \frac{1}{m} \sum_{j \neq j_0} p_j + p_{j_0} = \frac{1}{m} \sum_j p_j + \frac{m-1}{m} p_{j_0}$$

On the other hand, the optimal makespan $C^*_{\max}$ is larger than both p_{j_0} and $\frac{1}{m} \sum_j p_j$, which yields the desired result. Moreover, this ratio is tight as is shown by the following instance: Let $n = m(m-1)+1$ and denote by $j_1, j_2, ..., j_n$ the jobs ranked by LS. Assume that $p_{j_1} = p_{j_2} = ... = p_{j_{n-1}} = 1$ and $p_{j_n} = m$. Clearly, $C_{\max}(LS) = 2m-1$ and $C^*_{\max} = m$.

By applying a simple pairwise interchange to the list schedule, this worst-case ratio is improved to $2 - \frac{2}{m+1}$ (Graham et al. 1979). Achugbue and Chin (1981) proved that if $p_{\max} \leq 3p_{\min}$, then the worst-case ratio of a list schedule is

$$\begin{cases} \frac{5}{3} & \text{for } m = 3, 4 \\ \frac{17}{10} & \text{for } m = 5 \\ 2 - \frac{1}{3\left\lfloor \frac{m}{3} \right\rfloor} & \text{for } m \geq 6 \end{cases}$$

and if $p_{\max} \leq 2p_{\min}$, this ratio is

$$\begin{cases} \frac{3}{2} & \text{for } m = 2, 3 \\ \frac{5}{3} - \frac{1}{3\left\lfloor \frac{m}{2} \right\rfloor} & \text{for } m \geq 4 \end{cases}$$

where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x, and $p_{\min}$ and $p_{\max}$ denote the minimum and maximum processing time, respectively. Mokotoff et al. (2001) proposed new list schedules for the $P||C_{\max}$ and showed that they perform very well.

The accuracy of any list schedule is intimately related to the way the list is made. Several dispatching rules have been proposed in the literature

(Panwalkar and Iskander, 1977; Haupt, 1989). One of the most effective and yet simplest ones for the $P||C_{\max}$ is the so called *Longest Processing Time (LPT)* rule. It ranks the jobs according to the nonincreasing order of their processing times. The worst-case ratio of the LPT is $\frac{4}{3} - \frac{1}{3m}$ (Graham, 1969). Ibarra and Kim (1977) proved that this ratio is equal to $1 + \frac{2(m-1)}{n}$ if $p_{\min} \geq \frac{2(m-1)}{n} p_{\max}$. Moreover, Frenk and Rinnooy Kan (1987) proved that LPT is asymptotically an excellent algorithm (that is, for a large number of jobs, it yields optimal or near-optimal solution).

LPT continued to be the best algorithm until Coffman et al. (1978) presented the *MultiFit* (MF) algorithm. This algorithm takes into account the duality existing between the $P||C_{\max}$ and the bin packing problem (as can be seen in page 60). The idea behind MF is to use bisection search for finding the smallest capacity that a set of m bins can have and still accommodate all jobs when the jobs are sorted in nonincreasing order of their processing times and each job is placed into the first bin into which it will fit. Coffman et al. (1978) showed that if k packing attempts are performed, then MF runs in $O(n \log n + kn \log m)$ time and its worst-case ratio is

$$\begin{cases} \frac{8}{7} + 2^{-k} & \text{for } m = 2 \\ \frac{15}{13} + 2^{-k} & \text{for } m = 3 \\ \frac{20}{17} + 2^{-k} & \text{for } m = 4, 5, 6, 7 \\ c + 2^{-k} \text{ where } c \in [1.17, 1.22] & \text{for } m \geq 8 \end{cases}$$

For $m \geq 8$, the value of c has been subsequently proved to be less than 1.20 (Friesen, 1984). Then, Yue (1990) showed that c is actually equal to $\frac{13}{11}$. Friesen and Langston (1986) presented a refined version of MF which worst-case ratio is $\frac{72}{61} + 2^{-k}$. Lee and Massey (1988) proposed an interesting combination of MF and LPT. In the particular case of two machines, this combined heuristic improves the worst-case ratio of both MF and LPT. Indeed, it yields a ratio of $\min\left\{\frac{7}{6}, (\frac{10}{9} + \frac{2^{-k+1}}{6})\right\}$. Riera et al. (1996) proposed a new LPT-based algorithm which has been empirically proven to be very competitive with MF.

It is worth noting that Graham (1969) proposed another kind of constructive heuristics for the $P||C_{\max}$. It consists in scheduling the k largest jobs optimally then schedule the remaining jobs arbitrarily.. The worst-case ratio of this algorithm is $1+\frac{1-\frac{1}{m}}{1+\left\lfloor\frac{k}{m}\right\rfloor}$. Unfortunately, its running time is in $O(n^{km})$. Sahni (1976) improved this result by devising a similar algorithm which runs in $O(n(n^2k)^{m-1})$ time and has a worst-case ratio of $1+\frac{1}{k}$. Hochbaum and Shmoys (1987) proposed a polynomial approximation scheme using an entirely different approach. It is based on the so-called dual-approximation scheme which searches for an optimal solution from the set of super-optimal, but infeasible, solutions.

Significantly less is known about approximation algorithms for the $P|r_j, q_j|C_{\max}$. Indeed, no such algorithm has been proposed since Jackson (1955) proposed his well-known Jackson's rule. It consists in scheduling the available job with the largest tail at the earliest available machine. Gusfield (1984) proved that the maximum deviation of Jackson's schedule from the optimum is less than $\frac{2m-1}{m}\max_{j\in J} p_j$. Carlier (1987) improved this deviation for the case where $\max_{j\in J} p_j \leq 2m$ by providing the maximal deviation of $2(\max_{j\in J} p_j - 1)$. In Chapter 5, we present a new variant of Jackson's algorithm and show that it exhibits a better worst-case performance (cf. page 85).

Many of the above results can be generalized to the uniform machine model. The first work in this area is due to Liu and Liu (1974) who showed that the performance of list schedules on uniform machines is significantly worse than on identical machines. Indeed, the worst-case ratio of such algorithms is $1+\frac{s_{\max}}{s_{\min}}-\frac{s_{\max}}{\sum s_i}$, where s_i denotes the speed of machine i, and $s_{\max}$ and $s_{\min}$ denote the maximum and minimum machine speed, respectively. Morrison (1988) showed that the LPT has a worst-case ratio equal to $\max\left\{\frac{s_{\max}}{2s_{\min}}, 2\right\}$.

Instead of assigning the next job in the list on the first available machine, another implementation of list schedules on uniform machines consists in scheduling the next job in the list on the machine on which it will finish earliest. Cho and Sahni (1980) proved that this implementation improves the worst-case ratio of list schedules to $\frac{1+\sqrt{5}}{2}$ for $m=2$ and to $\frac{1+\sqrt{2m-2}}{2}$ for $m\geq 3$. Gonzalez et al. (1977) showed that an analogous generalization of

LPT rule yields a worst-case ratio of $\frac{1+\sqrt{17}}{4}$ for $m = 2$, and $2 - \frac{2}{m+1}$ for $m \geq 3$. Dobson (1984) improved the latter result by showing that the ratio lies in the interval $[1.512, 1.583]$ for $m \geq 3$. Friesen (1987) proved that this ratio is at least 1.52.

Friesen and Langston (1983) extended the MultiFit approach to the uniform model. They proved that, if the bins are ordered in increasing size for each of the k iterations of the binary search, the worst-case ratio of MF is $1.4 + 2^{-k}$. Chen (1991) tightened this ratio to $1.382 + 2^{-k}$. For $m = 2$, Burkard and He (1998) showed that MF has a worst-case ratio of $\frac{\sqrt{6}}{2} + 2^{-k}$. Moreover, they proved that MF combined with LPT improves this ratio to $\frac{\sqrt{2}+1}{2} + 2^{-k}$.

Unrelated parallel machine problems are perceived to be significantly harder than uniform machine problems. Horowitz and Sahni (1976) developed a dynamic-programming-based heuristic for the $R||C_{\max}$. Davis and Jaffe (1981) proposed a variant of list schedules for the unrelated machines model which worst-case ratio is $2.5\sqrt{m} + 1 + \frac{1}{2\sqrt{m}}$. Potts (1985) proposed a two-stage algorithm which provides a makespan which is no more than twice the optimum. The first stage uses linear programming to schedule at least $n-m+1$ jobs. The second stage schedules the remaining $m-1$ jobs using an enumerative method. In the particular case of two machines, this approach has a worst-case ratio of $\frac{3}{2}$ and runs in linear time. Lenstra et al. (1990) and Mokotoff and Jimeno (2002) extended this approach by combining partial schedule enumeration and linear programming. Martello et al. (1997) presented a quite sophisticated approximation algorithm for the $R||C_{\max}$. Lancia (2000) proposed an $O(n \log n)$ heuristic for the two-unrelated machines problem with heads and tails.

2.3.2 Improvement algorithms

These approximate algorithms are based upon the local search in a neighborhood. They take a feasible solution as starting-point and intend to improve it by iterative small changes. This iterative improvement can be achieved by means of many different processes. In particular, metaheuristics such as simulated annealing, tabu search and genetic algorithms can be used. From the moment of its introduction by Glover (1989; 1990), tabu search is probably the most tested local search concerning scheduling problems (Barnes et

al., 1995; Glover, 1996). Unlike the two other mentioned metaheuristics, genetic algorithms start with a set of solutions instead of only one. Although, research on metaheuristics for scheduling problems is quite extensive (Dorn et al. 1996), applications to parallel machine problems are relatively scarce.

Hübscher and Glover (1994) presented a tabu search approach to the $P||C_{\max}$. They introduced a diversification strategy which improved the quality of the derived solutions. Their algorithm was outperformed by the tabu search algorithms developed by Thesen (1998). Brucker et al. (1996, 1997) improved the neighborhood of $P||C_{\max}$ by constructing a neighborhood on the set of all locally optimal solutions. Jozefowska et al. (1998) combined linear programming and local search methods (tabu search, simulated annealing and genetic algorithms) to solve $P||C_{\max}$. They compared the performance of the three metaheuristics and showed that the tabu search is preferable, finding the largest number of optimal solutions and showing the smallest deviation for all the problem sizes. Scutella et al. (2000) introduced a new local search technique for the $P||C_{\max}$, whose neighborhood structure is based on multiple exchanges of jobs among machines. They showed that the proposed algorithm could derive rapid and satisfactory solutions. It is worth noting that an interesting improvement algorithm for the $P||C_{\max}$ has been proposed by Ho and Wong (1995). It breaks the original m-machine problem into a set of two-machine problems and optimally solves them repeatedly.

Several metaheuristics have been developed to deal with $R||C_{\max}$. Van de Velde (1993) presented an algorithm based on iterative local search, where the search direction is guided by surrogate multipliers. Its performance is better than all the previously published approximation algorithms. Later on, Glass et al. (1994) compared the relative performance of the three aforementioned metaheuristics on the $R||C_{\max}$. They found that the performance of standard genetic algorithms is poor if no other heuristic or some problem-specific features are incorporated. However, they showed that genetic algorithms combined with a constructive algorithm are comparable with the developed simulated annealing and tabu search algorithms. For these three algorithms, tabu search generates slightly better solutions in a short time, while genetic algorithms and simulated annealing improve as the time limit increases. Piersma and van Dijk (1996) introduced an efficient new method that produced solutions which are competitive to those provided by Glass et al. (1994). Also, they showed that a tabu search algorithm implemented with their efficient neighborhood search strategy performs better than general local search algorithms. A very effective tabu search algorithm using hashing to control the tabu restrictions of the search process has been pre-

sented by Srivastava (1998). Sourd (2001) presented two algorithms based on large neighborhood improvement procedures. One is based upon a partial and heuristic exploration of a search tree, and the other one is based on the duality approach of van de Velde (1993).

2.4 Extensions of standard parallel machine scheduling problems

In this section, we mention some of the most important variants of standard parallel machine scheduling problems and provide further references for the interested reader.

2.4.1 Limited machine availabilities

The basic scheduling model assumes that all machines are continuously available for processing throughout the planning horizon. This assumption does not apply if certain maintenance operations are planned during the same time horizon. Ullman (1975) was the first to study the parallel machine problem with machine time intervals. He proved by a reduction to the 3-partition problem that the problem is strongly $\mathcal{NP}$-hard. Schmidt (1988) investigated the problem of finding a preemptive schedule on identical parallel machines with job release dates or deadlines and machine availability time intervals. He showed that the problem can be solved in $O(nm \log m)$ time. Lee (1991) showed that, for the $P||C_{\max}$ with machine ready times, an appropriate modification of LPT rule yields a worst-case ratio of $\frac{1}{3}$. Kellerer (1998) improved this result by developing a dual approximation heuristic which worst-case ratio is $\frac{5}{4}$. Schmidt (2000) provides an exhaustive review of the main research developments related to the scheduling problems with machine availabilities.

2.4.2 Multiprocessor jobs

Some applications from the real life do not satisfy the assumption that no job can be processed on more than one machine at one time. For example, in multiprocessor systems one processor tests the others. So, in such models the jobs have to be attended by more than one machine simultaneously. According to the requirements one may distinguish two kinds of jobs. Some jobs must be processed by a given number of machines during a given processing time simultaneously. They are called nonmalleable. For malleable

jobs, their processing time is a non-increasing function of the number of machines available. Some approximation algorithms were recently developed by Ludwing and Tiwari (1994), Turek et al. (1994), Schwiegelshohn et al. (1997). Blazewicz et al. (1994a) and Drozdowski (1996) provide surveys for multiprocessor jobs problems.

2.4.3 Communication delays

Since the development of distributed memory computers, the study of scheduling problems with communication delays between precedence-related jobs assigned to different machines became a continuously increasing research area. In these problems, an additional cost is incurred if two successive jobs are not processed on the same machine. Veltman et al. (1990) and Chrétienne and Picouleau (1995) surveyed this class of problems. Most of the research effort has been devoted to develop heuristics (Hoogeveen et al., 1994; Hanen and Munier, 1995; Möhring et al., 1996 ;Munier and König, 1997).

Chapter 3

Mathematical properties and a preprocessing algorithm for the $P|r_j, q_j|C_{\max}$

In this chapter, we study some mathematical properties of the $P|r_j, q_j|C_{\max}$ and we present a preprocessing algorithm which consistently simplifies the problem by reducing the number of unscheduled jobs.

3.1 Mathematical formulation

A mixed-integer programming formulation of the $P|r_j, q_j|C_{\max}$ is described as follows. Let UB denote an upper bound on the optimal makespan. Consider the following decision variables:

- $C_{\max}$: the maximum completion time
- t_j : the starting time of job j
- $x_{jh} = \begin{cases} 1 & \text{if job } j \text{ is processed by machine } h \\ 0 & \text{otherwise} \end{cases}$
- $y_{jkh} = \begin{cases} 1 & \text{if job } j \text{ precedes job } k \text{ on machine } h \\ 0 & \text{otherwise} \end{cases}$

The $P|r_j, q_j|C_{\max}$ can be formulated as a mixed linear programming in the following way:

$$
\begin{cases}
Min\ C_{\max} & & \\
C_{\max} \geq t_j + p_j + q_j & j = 1, ..., n & (1) \\
t_j \geq r_j & j = 1, ..., n & (2) \\
\sum_{h=1}^{m} x_{jh} = 1 & j = 1, ..., n & (3) \\
y_{jkh} + y_{kjh} \leq x_{jh} & j, k = 1, ..., n; h = 1, ..., m & (4) \\
y_{jkh} + y_{kjh} \geq x_{jh} + x_{kh} - 1 & j, k = 1, ..., n; h = 1, ..., m & (5) \\
t_j + p_j x_{jh} \leq t_k + UB(1 - y_{jkh}) & j, k = 1, ..., n; h = 1, ..., m & (6) \\
t_k + p_k x_{kh} \leq t_j + UB(1 - y_{kjh}) & j, k = 1, ..., n; h = 1, ..., m & (7) \\
C_{\max} \geq 0; t_j \geq 0; x_{jh} \in \{0, 1\}; y_{jkh} \in \{0, 1\} & j, k = 1, ..., n; h = 1, ..., m & (8)
\end{cases}
$$

The constraints (1) stem from the definition of $C_{\max}$ which has to be greater than the finishing time of all the jobs. The constraints (2) ensure that each job cannot start processing before its release date, whereas constraints (3) ensure that a job has to be processed by exactly one machine. Finally, constraints (4)-(7) express the nonpreemption and non-overlapping constraints. Note that if jobs j and k are not processed on the same machine h, then constraints (4) lead to $y_{jkh} = y_{kjh} = 0$, and constraints (5), (6) and (7) become redundant.

3.2 Symmetry of the problem

In this section, we point out an interesting property of the $P|r_j, q_j|C_{\max}$ which is its symmetry. Formally, this property can be stated as follows:

Observation 3.1

Given a feasible $P|r_j, q_j|C_{\max}$ schedule, the symmetric schedule obtained by

reversing the roles of heads and tails has the same makespan as the original one.

Proof. Assume that a feasible schedule σ with makespan $C_{\max}$ is constructed for a $P|r_j, q_j|C_{\max}$ problem. It is assumed with no loss of generality that at least one job starts processing at its release date in σ. Denote by t_j the starting time of job j in σ and by $j+1$ its immediate successor on the same machine. We have :

$$\begin{cases} t_j \geq r_j & \text{for all } j \in J \\ t_{j+1} \geq t_j + p_j & \text{for all } j \in J \\ C_{\max} = \max\limits_{j \in J} (t_j + p_j + q_j) \end{cases}$$

A feasible schedule σ' with the same makespan can be easily obtained for the symmetric problem by keeping the assignments of jobs on machines and setting starting times equal to $t'_j = C_{\max} - t_j - p_j$, for each job j. Job j is therefore scheduled just after job $j+1$ on the same machine in σ'. We have to prove that σ' is a feasible schedule for the symmetric problem and that its makespan is equal to $C_{\max}$. It suffices to show that :

$$\begin{cases} t'_j \geq q_j & \text{for all } j \in J \\ t'_j \geq t'_{j+1} + p_{j+1} & \text{for all } j \in J \\ C_{\max} = \max\limits_{j \in J} (t'_j + p_j + r_j) \end{cases}$$

The first inequality is obvious as $C_{\max} \geq t_j + p_j + q_j$ for all $j \in J$. Since we have

$$t_{j+1} \geq t_j + p_j \qquad \text{for all } j \in J$$

then

$$C_{\max} - t'_{j+1} - p_{j+1} \geq C_{\max} - t'_j - p_j + p_j \qquad \text{for all } j \in J$$

which leads to

$$t'_j \geq t'_{j+1} + p_{j+1} \qquad \text{for all } j \in J$$

Let $C'_{\max}$ denote the makespan of σ'. We have

$$C'_{\max} = \max_{j \in J} (t'_j + p_j + r_j) = \max_{j \in J} (C_{\max} - t_j + r_j) = C_{\max} + \max_{j \in J} (r_j - t_j)$$

Since $t_j \geq r_j$ for every $j \in J$, and there is necessarily a job which starts processing at its release date in σ, then $\max_{j \in J} (r_j - t_j) = 0$. ■

In the sequel, we will refer to the original problem as the *Forward* problem and to the symmetric one as the *Backward* problem. An immediate consequence of the above observation is the following result.

Corollary 3.1

The Forward and the Backward problems have the same optimal makespan

3.3 Equivalence with the maximum lateness problem

It is worth noting that the $P|r_j, q_j|C_{\max}$ problem can be presented in a slightly different form. For that purpose, consider the following practical scheduling problem. A set of jobs need to be processed on m identical parallel machines. Each job has a ready date r_j and a due date d_j, which represents the date the job is promised to the customer. The lateness L_j of job j is defined as the amount of time $p_j + t_j - d_j$. The maximum lateness is defined as $L_{\max} = \max_j L_j$. Assume that the scheduler is concerned with minimizing the maximum lateness. The resulting problem is denoted $P|r_j|L_{\max}$. We can make the following observation:

Observation 3.2

$P|r_j|L_{\max}$ *is equivalent to* $P|r_j, q_j|C_{\max}$

Proof. Consider an instance of $P|r_j|L_{\max}$, let $D = \max_j d_j$, and $q_j = D - d_j$ then we have

$$\begin{aligned} L_{\max} &= \max_j (p_j + t_j - d_j) \\ &= \max_j (p_j + t_j + q_j - D) \\ &= \max_j (p_j + t_j + q_j) - D \\ &= C_{\max} - D \end{aligned}$$

Conversely, given an instance of $P|r_j, q_j|C_{\max}$, an equivalent $P|r_j|L_{\max}$ instance can be obtained by setting for any job j, a due date $d_j = C - q_j$, where C is an upper bound of the optimal makespan. ■

An immediate consequence of observation 3.2 is that an algorithm for $P|r_j, q_j|C_{\max}$ solves also the $P|r_j|T_{\max}$.

3.4 Problem complexity

In this section, we prove that the $P|r_j, q_j|C_{\max}$ problem is strongly $\mathcal{NP}$-hard (that is even a pseudo-polynomial algorithm is not likely to exist for solving the problem). Indeed, $P|r_j, q_j|C_{\max}$ is a generalization of the classical $1|r_j, q_j|C_{\max}$, which is known to be $\mathcal{NP}$-hard in the strong sense (Garey and Johnson, 1979). A formal proof of the strong $\mathcal{NP}$-hardness of $P|r_j, q_j|C_{\max}$ is described below. The proof is based on a reduction of the equivalent $P|r_j|L_{\max}$ to the 3-partition problem.

The 3-partition problem (Garey and Johnson, 1979) is a notorious strongly $\mathcal{NP}$-hard decision problem which can be described as follows: Let N and B be two positive integers, and let $A = \{a_1, a_2, ..., a_{3N}\}$ be a set of positive integers such that $\sum_{i=1}^{3N} a_i = NB$ and $\frac{B}{4} < a_i < \frac{B}{2}$ for $i = 1, 2, ..., 3N$. Does there exist a partition of A into three element sets $\{A_1, A_2, ..., A_N\}$ such that $\sum_{a_j \in A_i} a_j = B$ for $i = 1, ..., N$?

Let J be the set of $4N - 1$ jobs partitioned into two types :

Type 1 : $r_j = jB + (j-1), p_j = 1, d_j = jB + j$ for $j = 1, ..., N-1$

Type 2 : $r_j = 0, p_j = a_{j-N+1}, d_j = NB + (N-1)$ for $j = N, ..., 4N-1$

Let $\overline{A} = \{A_1, A_2, ..., A_m\}$ where $A_i = A$ $(i = 1, 2, ..., m)$ and assume that we have m sets of jobs which data are identical to those of J. Denote by $\overline{J}$ the set containing the resulting $(4N-1)m$ jobs.

Consider the m machines instance of $P|r_j|L_{\max}$ defined on $\overline{J}$. A schedule with $L_{\max} \leq 0$ exists if and only if every job j of type 1 is processed between r_j and $d_j = r_j + p_j$ (see figure 3.1). This can be done if and only if jobs of type 2 can be partitioned over the mN intervals of length B. That is, there

exists a partition of $\overline{A}$ into three set elements $\{\overline{A}_1, \overline{A}_2, ..., \overline{A}_{mN}\}$ such that $\sum_{a_j \in \overline{A}_i} a_j = B$ for $i = 1, ..., mN$. This can be done if and only if the 3-partition problem has a solution. Thus, the $P|r_j, q_j|C_{\max}$ is $\mathcal{NP}$-hard in the strong sense.

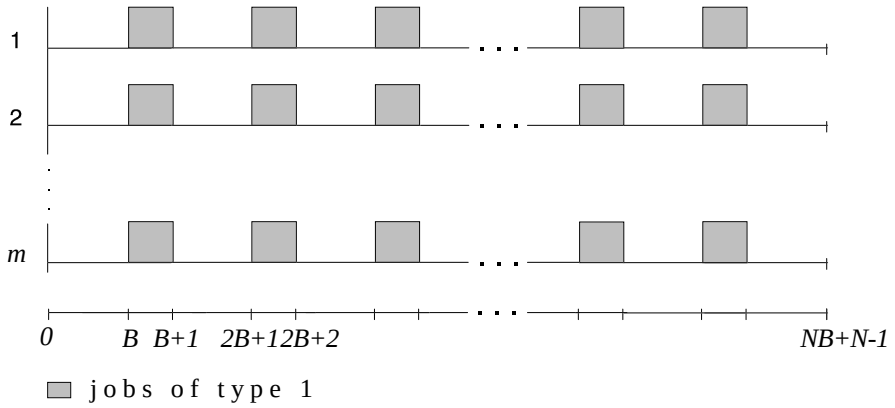


Figure 3.1. $P|r_j, q_j|C_{\max}$ is strongly $\mathcal{NP}$-hard

3.5 A preprocessing algorithm

In this section, we introduce a preprocessing algorithm for the $P|r_j, q_j|C_{\max}$ which aims at simplifying the problem by reducing the number of jobs to be scheduled. Several authors have shown that preprocessing often improves the efficiency of branch-and-bound algorithms (see for instance Hoffman and Padberg, 1991; Savelsbergh, 1994).

3.5.1 Selection rules and algorithm

The following lemma provides a sufficient condition which allows to identify the starting time of particular jobs in an optimal schedule.

Lemma 3.1

Assume that the jobs of J ($|J| > m$) *are ranked in nondecreasing order of their release dates. Denote by* $j_{(k)}$ *the job which has the* k^{th} *smallest release date. Let* $J_{r,m} = \{j_{(1)}, j_{(2)}, ..., j_{(m)}\}$ *and* $j_0 \in J_{r,m}$ *such that* $r_{j_0} + p_{j_0} = \min_{j \in J_{r,m}} (r_j + p_j)$.

Assume that the following condition is satisfied :

$$r_{j_0} + p_{j_0} \leq r_{j_{(m+1)}} \tag{3.1}$$

then :

$$C^*_{\max}(J) = \max\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J\backslash\{j_0\})\}$$

where $C^*_{\max}(S)$ denotes the optimal makespan for the $P|r_j, q_j|C_{\max}$ problem defined for the set of jobs S.

Proof. Clearly, we have

$$\max\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J\backslash\{j_0\})\} \leq C^*_{\max}(J)$$

Consider the $P|r_j, q_j|C_{\max}$ defined for $J\backslash\{j_0\}$. In any optimal schedule for this problem, there is necessarily one machine m_0 which is idle from time zero to $r_{j_{(m+1)}}$. Then, scheduling j_0 on m_0 at time r_{j_0} yields a feasible schedule, for the $P|r_j, q_j|C_{\max}$ defined for J, with makespan equal to $\max\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J\backslash\{j_0\})\}$. ■

An immediate consequence of this lemma is that if a job j_0 satisfies condition (3.1), then there exists an optimal schedule where its starting time is fixed at r_{j_0}. Therefore, j_0 can be removed from the set of jobs to be scheduled. We recursively apply Lemma 3.1 to the reduced jobset $J\backslash\{j_0\}$, until there is no job which satisfies condition (3.1) or there are no more than m unscheduled jobs.

An immediate consequence of the above lemma can be derived using the symmetry of the $P|r_j, q_j|C_{\max}$.

Lemma 3.2

Assume that the jobs of J ($|J| > m$) are ranked in nondecreasing order of their delivery times. Denote by $j_{(k)}$ the job which has the k^{th} smallest delivery time. Let $J_{q,m} = \{j_{(1)}, j_{(2)}, ..., j_{(m)}\}$ and $j_0 \in J_{q,m}$ such that $q_{j_0} + p_{j_0} = \min_{j \in J_{q,m}} (q_j + p_j)$.

Assume that the following condition is satisfied :

$$q_{j_0} + p_{j_0} \leq q_{j_{(m+1)}} \tag{3.2}$$

then :

$$C^*_{\max}(J) = \max\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J\backslash\{j_0\})\}$$

Thus, by performing a preprocessing based on condition (3.2), it is possible to remove a job j_0 satisfying this condition from the set of unscheduled jobs. Then, this preprocessing rule will be applied recursively to the jobset $J\backslash\{j_0\}$ until there is no job which satisfies condition (3.2) or there are no more than m unscheduled jobs. A maximal number of jobs can be fixed by iteratively applying the two symmetric preprocessing rules. Therefore, the problem can be solved on a reduced jobset, denoted by $\overline{J}$. Denote by J_R and J_Q the sets of jobs removed according to the preprocessing rules based on conditions (3.1) and (3.2), respectively. Lemmas 3.1 and 3.2 prove that there is an optimal schedule in which the jobs of J_R precede those of $\overline{J}$, and the jobs of J_Q succeed those of $\overline{J}$. A formal description of the preprocessing algorithm is given below.

Preprocessing Algorithm

Step 0. *Set* $J_R = \emptyset, J_Q = \emptyset$ *and* $\overline{J} = J$.

Step 1. Test on condition (3.1)

1.1. *If* $\left|\overline{J}\right| \leq m$, *then stop.*
1.2. *If no job* $j_0 \in \bar{J}$ *satisfies condition (3.1), then go to step 2. Else, set* $\overline{J} = \overline{J}\backslash\{j_0\}$, $J_R = J_R \cup \{j_0\}$ *and go to step 1.1.*

Step 2. Test on condition (3.2)

2.1. *If* $\left|\overline{J}\right| \leq m$, *then stop.*
2.2. *If no job* $j_0 \in \bar{J}$ *satisfies condition (3.2), then go to step 3. Else, set* $\overline{J} = \overline{J}\backslash\{j_0\}$, $J_Q = J_Q \cup \{j_0\}$ *and go to step 2.1.*

Step 3. *If no job is removed in step 2, then stop. Else, go to step 1.*

Ranking jobs according to r_j, $r_j + p_j$, q_j, and $p_j + q_j$ requires $O(n \log n)$ time. There are at most $n - m$ iterations. Thus, the complexity of the preprocessing algorithm is $O(n \log n)$.

Example 3.1. Consider the 9 job-2 machine instance defined by Table 3.1. The main steps of the preprocessing algorithm are the following.

Iteration 1.

Step 1. $J_{r,m} = \{9, 1\}$, $j_0 = 9$ and $j_{(m+1)} = 3$. Since $r_{j_0} + p_{j_0} = 2$ and $r_{j_{(m+1)}} = 2$, then j_0 satisfies condition (3.1). Therefore, job 9 is removed from

$\bar{J}$ and added to J_R. Now, $J_{r,m} = \{1, 3\}$, $j_0 = 1$ and $j_{(m+1)} = 8$. Condition (3.1) is not satisfied by j_0 since $r_{j_0} + p_{j_0} = 3$ and $r_{j_{(m+1)}} = 2$.

Step 2. $J_{q,m} = \{8, 6\}$, $j_0 = 8$ and $j_{(m+1)} = 7$. Since $q_{j_0} + p_{j_0} = 7$ and $q_{j_{(m+1)}} = 7$, then j_0 satisfies condition (3.2). Therefore, job 8 is removed from $\bar{J}$ and added to J_Q. Similarly, jobs 6 and 7 are successively transferred from $\bar{J}$ to J_Q.

Iteration 2.

Step 1. $J_{r,m} = \{1, 3\}$, $j_0 = 1$ and $j_{(m+1)} = 2$. Since $r_{j_0} + p_{j_0} = 3$ and $r_{j_{(m+1)}} = 4$, then j_0 satisfies condition (3.1). Therefore, job 1 is removed from $\bar{J}$ and added to J_R. The next job satisfying condition (3.1) is job 2 which is moved to J_R.

Step 2. $J_{q,m} = \{4, 5\}$, $j_0 = 5$, $j_{(m+1)} = 3$ and $q_{j_0} + p_{j_0} = 16 > q_{j_{(m+1)}} = 15$. No job is added to J_Q. Therefore, the algorithm stops.

Hence, the preprocessing algorithm outputs: $J_R = \{9, 1, 2\}$, $J_Q = \{8, 6, 7\}$, and $\bar{J} = \{3, 4, 5\}$.

j	1	2	3	4	5	6	7	8	9
r_j	1	4	2	7	8	12	14	2	0
p_j	2	3	7	3	5	6	4	5	2
q_j	14	14	15	14	11	2	7	2	1

Table 3.1. Data of the 9 job - 2 machine instance of example 3.1

3.5.2 Global Schedule Construction Algorithm

After performing the preprocessing algorithm, the set J is partitioned into three subsets : J_R, J_Q and $\overline{J}$. A consequence of Lemmas 3.1 and 3.2 is that :

$$C^*_{\max}(J) = \max\left\{C^*_{\max}(\overline{J}), \max_{j \in J_R \cup J_Q} (r_j + p_j + q_j)\right\}$$

Assume that we solve the problem on the jobset $\overline{J}$, and that σ is a schedule of $\overline{J}$ with makespan equal to $C_{\max}(\overline{J})$. The algorithm described below integrates the jobs of J_R and J_Q in the schedule σ in order to construct a global schedule for the entire jobset J. We refer to this algorithm as the Global Schedule Construction Algorithm ($GSCA$). Let $u_i(\sigma)$ denote the availability time of the machine m_i $(i = 1, ..., m)$ in the schedule σ, and let σ^{-1} be the symmetric schedule of σ.

Global Schedule Construction Algorithm ($GSCA$).

Step 1.

1.1 *If* $J_Q = \emptyset$, *then go to step 2. Else, let* $j_0 \in J_Q$ *be the job such that* $p_{j_0} + q_{j_0} = \max_{j \in J_Q} (p_j + q_j)$.

1.2 *Let* m_0 *be the machine with availability time* $u_0 = \min_{i=1,\ldots,m} u_i(\sigma)$. *Schedule job* j_0 *on* m_0 *and set* $u_0 = \max(u_0, r_{j_0}) + p_{j_0}$.

1.3 *Set* $J_Q = J_Q \backslash \{j_0\}$ *and go to step 1.1.*

Step 2. *Set* $\sigma = \sigma^{-1}$.

Step 3.

3.1 *If* $J_R = \emptyset$, *then go to step 4. Else, let* $j_0 \in J_R$ *be the job such that* $r_{j_0} + p_{j_0} = \max_{j \in J_R} (r_j + p_j)$.

3.2 *Let* m_0 *be the machine with availability time* $u_0 = \min_{i=1,\ldots,m} u_i(\sigma)$. *Schedule job* j_0 *on* m_0 *and set* $u_0 = \max(u_0, q_{j_0}) + p_{j_0}$.

3.3 *Set* $J_R = J_R \backslash \{j_0\}$ *and go to step 3.1.*

Step 4. *Set* $\sigma = \sigma^{-1}$.

Lemma 3.3

Given J_R, J_Q *and* σ *with makespan* $C_{\max}(\overline{J})$, *the GSCA yields a schedule with makespan equal to*

$$C_{\max}(J) = \max \left\{ C_{\max}(\overline{J}), \max_{j \in J_R \cup J_Q} (r_j + p_j + q_j) \right\}$$

Proof. First, we prove that if a job j_1 is removed before job j_2 from $\bar{J}$ according to condition (3.2), then we have $p_{j_1} + q_{j_1} \leq p_{j_2} + q_{j_2}$. Indeed, we have $p_{j_1} + q_{j_1} = \min_{j \in \bar{J}_{q,m}} (p_j + q_j)$ and $p_{j_1} + q_{j_1} \leq q_{j_{(m+1)}}$.

On the other hand, we have

$$\begin{aligned} p_{j_2} + q_{j_2} &= \min_{j \in \bar{J}_{q,m} \setminus \{j_1\} \cup \{j_{(m+1)}\}} (p_j + q_j) \\ &= \min\{ \min_{j \in \bar{J}_{q,m} \setminus \{j_1\}} (p_j + q_j),\, p_{j_{(m+1)}} + q_{j_{(m+1)}} \} \\ &\geq p_{j_1} + q_{j_1}. \end{aligned}$$

Consequently, the last job removed from $\bar{J}$ and put in J_Q is the job j_0 with maximal $p_j + q_j$, and we have $p_{j_0} + q_{j_0} \leq q_{j_{(m)}}$, where $j_{(m)}$ is the job of $\bar{J}$ with the m^{th} smallest tail. If j_0 is scheduled on m_0 (the machine with minimal availability time u_0) at the last position, then two cases are possible:

1- j_0 starts at r_{j_0} : Its completion time will be therefore equal to $r_{j_0} + p_{j_0} + q_{j_0} \leq \max_{j \in J_Q} (r_j + p_j + q_j)$.

2- j_0 starts at u_0 : Note that there are at least m jobs in $\bar{J}$. Thus, there must exist a machine m_k in σ on which the last scheduled job j_k has a tail greater than or equal to $q_{j_{(m)}}$. We have

$$C_{\max}(\bar{J}) \geq u_k(\sigma) + q_{j_k} \geq u_0 + q_{j_{(m)}} \geq u_0 + p_{j_0} + q_{j_0}$$

Therefore, in both cases, fixing j_0 on machine m_0 yields a schedule with makespan not greater than $C_1 = \max \left\{ C_{\max}(\bar{J}), \max_{j \in J_Q} (r_j + p_j + q_j) \right\}$. By recursion, it is possible to schedule the jobs of J_Q according to non-increasing $p_j + q_j$ in such a way that the maximum completion time of the obtained schedule is C_1. By symmetry, the jobs of J_R can be scheduled in the same way. The maximum completion time of the resulting schedule is therefore

$$C_{\max}(J) = \max \left\{ C_{\max}(\overline{J}), \max_{j \in J_R \cup J_Q} (r_j + p_j + q_j) \right\}. \blacksquare$$

Example 3.1 :(continued) Consider the instance defined by Table 3.1. We have $J_R = \{9, 1, 2\}$, $J_Q = \{8, 6, 7\}$ and $\overline{J} = \{3, 4, 5\}$. Consider the schedule of $\overline{J}$ with makespan equal to 25 depicted in Figure 3.2. $GSCA$ yields a schedule of J, depicted in Figure 3.3, with makespan equal to 25.

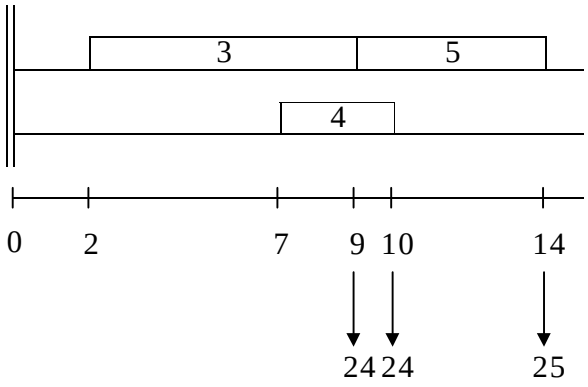


Figure 3.2. Schedule of $\bar{J}$

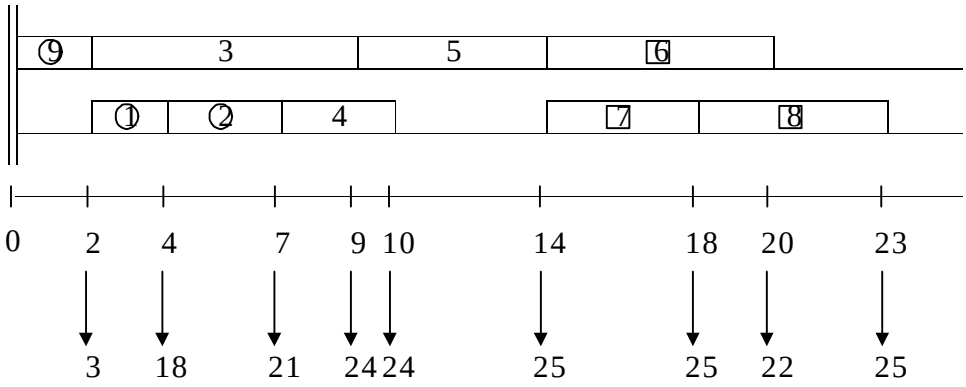


Figure 3.3. Schedule of J obtained after application of $GSCA$

An immediate consequence of Lemma 3.3 is the following corollary.

Corollary 3.2

If σ^* is an optimal schedule for $\overline{J}$ then $GSCA$ yields an optimal schedule for J.

Chapter 4

Lower bounds for the $P|r_j, q_j|C_{\max}$

In this chapter, we investigate new lower bounds for the $P|r_j, q_j|C_{\max}$ scheduling problem. Several new relaxation schemes, as well as new lifting procedures are derived. Extensive numerical experiments show that the proposed lower bounds consistently outperform the best existing ones.

In the sequel, the values $\bar{r}_j(J)$ and $\bar{q}_j(J)$ denote the j^{th} smallest release date and delivery time in J, respectively. We denote by $\lceil x \rceil$ the smallest integer that is larger than or equal to x, and by $\lfloor x \rfloor$ the largest integer that is smaller than or equal to x.

4.1 Lower bounds from the literature

4.1.1 Simple lower bounds

Obviously, no job j can finish its processing earlier than $r_j + p_j + q_j$. Thus, a simple lower bound is:

$$LB_0(J) = \max_{j \in J} (r_j + p_j + q_j)$$

LB_0 can be computed in $O(n)$ time. A second simple $O(n)$ lower bound given by Carlier (1987) is:

$$LB_1(J) = \min_{j \in J} r_j + \left\lceil \frac{1}{m} \sum_{j \in J} p_j \right\rceil + \min_{j \in J} q_j$$

This bound is based on the fact that all release dates and delivery times are assumed to be equal to $\min_{j\in J} r_j$ and $\min_{j\in J} q_j$, respectively.

Now, in any feasible schedule, a machine is necessarily idle from time zero to $\overline{r}_1(J)$, a second one from time zero to $\overline{r}_2(J)$, ..., and an m^{th} one from time zero to $\overline{r}_m(J)$. Similarly, a machine is idle from time $C^*_{\max}(J)-\overline{q}_1(J)$ to $C^*_{\max}(J)$, ..., and an m^{th} one from time $C^*_{\max}(J)-\overline{q}_m(J)$ to $C^*_{\max}(J)$. Moreover, the total amount of activity of the m machines is equal to $\sum_{j\in J} p_j$. By adding the minimal amounts of activity to those of inactivity, we obtain:

$$\overline{r}_1(J)+\overline{r}_2(J)+\cdots+\overline{r}_m(J)+\sum_{j\in J} p_j+\overline{q}_1(J)+\overline{q}_2(J)+\cdots+\overline{q}_m(J) \leq mC^*_{\max}(J)$$

This yields the following $O(n)$ lower bound proposed by Carlier (1987):

$$LB_2(J) = \left\lceil \frac{1}{m}\left(\overline{r}_1(J)+\cdots+\overline{r}_m(J)+\sum_{j\in J} p_j+\overline{q}_1(J)+\cdots+\overline{q}_m(J)\right)\right\rceil$$

Clearly, $LB_2(J)$ dominates $LB_1(J)$.

Interestingly, we prove that the preprocessing algorithm improves LB_2.

Proposition 4.1.

$LB_2(\overline{J}) \geq LB_2(J)$

Proof. It suffices to prove that each removal of a job $j_0 \in J_R$ from J improves the bound LB_2. By definition, job j_0 belongs to $J_{r,m}$, where $J_{r,m}$ is the set of jobs in J with the m smallest heads. Let $J'_{r,m}$, $J'_{q,m}$ and J' be the respective updated sets $J_{r,m}$, $J_{q,m}$ and J if job j_0 is removed from J. Then,

$$\sum_{j\in J'_{r,m}} r_j+\sum_{j\in J'} p_j = \sum_{j\in J_{r,m}} r_j - r_{j_0} + r_{j_{(m+1)}} + \sum_{j\in J} p_j - p_{j_0}$$

Since $r_{j_0}+p_{j_0} \leq r_{j_{(m+1)}}$, then

$$\sum_{j\in J'_{r,m}} r_j+\sum_{j\in J'} p_j \geq \sum_{j\in J_{r,m}} r_j+\sum_{j\in J} p_j$$

On the other hand, $\sum_{j\in J'_{q,m}} q_j \geq \sum_{j\in J_{q,m}} q_j$. Therefore,

$$LB_2(J\backslash\{j_0\}) \geq LB_2(J)$$

Consequently, $LB_2(J\backslash J_R) \geq LB_2(J)$. Similarly, it can be proven that $LB_2(J\backslash J_Q) \geq LB_2(J)$. By applying the last inequality to $J\backslash J_R$ we obtain

$$LB_2(\overline{J}) = LB_2((J\backslash J_R)\backslash J_Q) \geq LB_2(J\backslash J_R) \geq LB_2(J) \blacksquare$$

4.1.2 The adjusted lower bound

Vandevelde (1994) introduced a procedure which aims at enhancing the bound LB_2 by taking into account in a more effective way the heads and the tails. It may be described as follows: In the computation of LB_2, it is implicitly assumed that the m jobs with minimal heads are processed first on the machines while the m jobs with minimal tails are processed last. However, in a nonpreemptive schedule, a job can be both the first one and the last one to be processed on a machine only in the case where it is the only job processed on that machine. Let $LB_2^*(J, S, m)$ denote the lower bound LB_2 computed for the set of jobs J that must be scheduled on m machines such that it is forbidden to use both head and tail of any job in the set S.Assume that both head and tail of a job $j_0 \in J\backslash S$ are used in the computation of $LB_2^*(J, S, m)$. Then, the bound can be adjusted by taking the minimum value of the two bounds corresponding to the two following alternatives:

i) job j_0 is processed on a machine and all other jobs are processed on the other machines: a valid lower bound is

$$\max\{r_{j_0} + p_{j_0} + q_{j_0}; LB_2^*(J\backslash\{j_0\}, S, m-1)\}$$

ii) the head and the tail of j_0 cannot both be used in the computation of LB_2 : then, a valid lower bound is

$$LB_2^*(J, S \cup \{j_0\}, m)$$

Therefore, a recursive adjustment of $LB_2^*(J, S, m)$ is given by

$$LB_2^*(J, S, m) = \min \left\{ \begin{array}{c} \max\{r_{j_0} + p_{j_0} + q_{j_0}; LB_2^*(J\backslash\{j_0\}, S, m-1)\}; \\ \\ LB_2^*(J, S \cup \{j_0\}, m) \end{array} \right\}$$

Consequently, the bound LB_2 can be adjusted by computing $LB_2^*(J, S, m)$ using the above recursive formula. The resulting bound is referred to as the Adjusted Lower Bound and is denoted hereafter by $ALB_2(J)$.

Now, we describe how the bound $LB_2^*(J, S, m)$ can be computed. The problem is to determine two sets both of m jobs, denoted by R and Q, with

the restriction that $R \cap Q \nsubseteq S$, and such that $f(R,Q) = \sum_{j \in R} r_j + \sum_{j \in Q} q_j$ is minimized. Let j_r denote the job with maximal head in R, and j_q denote the job with maximal tail in Q. Denote by r and q the head and the tail of j_r and j_q, respectively. A necessary condition for the optimality of a given pair of sets (R,Q) is provided in the following:

Observation 4.1.

Assume that (R,Q) is optimal, then the two following conditions have to be satisfied:

i) For any job j such that $r_j \leq r$, we have

$$j \notin R \Longrightarrow j \in S \cap Q$$

ii) For any job j such that $q_j \leq q$, we have

$$j \notin Q \Longrightarrow j \in S \cap R$$

Proof. Let j_0 be a job such that $r_{j_0} \leq r$ and $j_0 \notin R$. Assume that $j_0 \notin S \cap Q^*$. Then, replacing any job in R by j_0 would yield a better or equal result, which contradicts the optimality of R. The second condition holds if we interchange heads and tails. ■

An immediate consequence of the above observation is the following corollary:

Corollary 4.1.

i) If $r_j \leq r$ and $q_j > q$, then $j \in R$

ii) If $r_j \leq r$, $q_j \leq q$ and $j \notin S$, then $j \in R$

iii) If $q_j \leq q$ and $r_j > r$, then $j \in Q$

iv) If $q_j \leq q$, $r_j \leq r$ and $j \notin S$, then $j \in Q$

Let us introduce the following notations:

$$R_1 = \{j \in J;\ r_j \leq r \text{ and } (q_j > q \text{ or } (q_j \leq q \text{ and } j \notin S))\}$$

$$Q_1 = \{j \in J;\ q_j \leq q \text{ and } (r_j > r \text{ or } (r_j \leq r \text{ and } j \notin S))\}$$

$S_{r,q} = \{j \in J;\ r_j \leq r,\ q_j \leq q \ and \ j \in S\}$

$n_r = |S_{r,q} \cap R|$

$n_q = |S_{r,q} \cap Q|$

$J_r = \{j \in J;\ r_j \leq r\}$

$J_q = \{j \in J;\ q_j \leq q\}$

Clearly, we have $|J_r| = |R_1| + |S_{r,q}|$ and $|J_q| = |Q_1| + |S_{r,q}|$. Also, Corollary 4.1 yields

$$\begin{cases} |R_1| + n_r = |R| = m \\ |Q_1| + n_q = |Q| = m \end{cases}$$

Moreover, according to Observation 4.1, any job of $S_{r,q}$ belongs either to R or to Q. That is, $n_r + n_q = |S_{r,q}|$. Consequently, we have

$$|J_r| + |J_q| = 2m + |S_{r,q}| \tag{4.1}$$

The optimal pair (R, Q) is computed by enumerating all the possible pairs of jobs (j_r, j_q) such that equation (4.1) is satisfied. For a given valid pair (j_r, j_q), the sets R and Q are constructed from J_r and J_q as follows. Firstly, we add the set R_1 and Q_1 to R and Q, respectively. Now, we describe how to determine the $m - |R_1| = n_r$ jobs of $S_{r,q}$ which have to be added to R. Let x_j be the decision variable defined as follows:

$$x_j = \begin{cases} 1 & \text{if } j \in R \\ 0 & \text{otherwise} \end{cases}$$

Since any job $j \in S_{r,q}$ belongs either to R or to Q, but not both, then its contribution to $f(R, Q)$ is $r_j x_j + q_j(1 - x_j)$. We have to find the n_r jobs of $S_{r,q}$ which yield the minimal total contribution. Formally, we have the following optimization problem:

$$\begin{cases} Min \sum\limits_{j \in S_{r,q}} (r_j - q_j)x_j + \sum\limits_{j \in S_{r,q}} q_j \\ \\ \sum\limits_{j \in S_{r,q}} x_j = n_r \\ \\ x_j \in \{0, 1\} \end{cases}$$

Clearly, an optimal solution of this problem consists in considering the n_r jobs of $S_{r,q}$ with minimal $r_j - q_j$. The set Q is completed with the remaining n_q jobs of $S_{r,q}$.

For the computation of $LB_2^*(J, S, m)$, we construct a list of jobs L_r sorted according to the increasing order of their heads, where ties are settled according to nondecreasing tails, and a list of jobs L_q sorted according to the increasing order of their tails, where ties are settled according to nondecreasing heads. Let a and b denote the indices of the job j_r and j_q in the lists L_r and L_q, respectively. Note that $a = |J_r|$ and $b = |J_q|$.

Actually, we only need to enumerate the possible values of a. Indeed, equation (4.1) yields the corresponding value of b in an optimal solution. If there are several values of b satisfying equation (4.1), we choose the largest value. Indeed, assume that two values b' and b'' ($b' < b''$) satisfy equation (4.1) for a given value of a. That is, for a given job j_r, there are two possible corresponding jobs j' and j'' such that $q_{j'} \leq q_{j''}$. If $q_{j'} < q_{j''}$, then $\{j \in J; q_j \leq q_{j'}\} \subset \{j \in J; q_j \leq q_{j''}\}$. Thus, the objective function $f(R, Q)$ is clearly minimized if we consider the job j''. If $q_{j'} = q_{j''}$, then by construction of L_q, we have $r_{j'} \leq r_{j''}$. Therefore, by choosing j'', we obtain the same value of $f(R, Q)$ than with j', but having a smaller or equal number of jobs in $R \cap Q$.

It suffices to take the lists L_r and L_q as the $2m$ jobs with minimal heads and tails, respectively. Indeed, as $n_{r,q} \leq \min(a, b)$, then equation (4.1) yields $m \leq a \leq 2m$ and $m \leq b \leq 2m$. Moreover, we need to create a list $L_{r,q}$ of the jobs in $L_r \cap L_q$ sorted according to the nondecreasing order of $r_j - q_j$. Each time we increase the value of a by one unit, we need to adjust the corresponding value of b, which requires $O(m)$ operations. Constructing the sets R and Q can be performed in $O(m)$ time since $L_{r,q}$ contains at most $2m$ jobs. Consequently, the bound $LB_2^*(J, S, m)$ requires $O(m^2)$ time.

In the computation of ALB_2, there are no more than 2^{2m} calls of $LB_2^*(J, S, m)$. Therefore, this bound can be computed in $O(2^{2m}m^2)$ time. The computation of $ALB_2(J)$ is illustrated by the following example.

Example 4.1. Consider the 7 job -2 machine instance defined by Table 4.1.

j	1	2	3	4	5	6	7
r_j	42	92	28	5	93	1	17
p_j	99	93	92	98	97	97	96
q_j	35	43	92	54	22	25	96

Table 4.1. Data of the 7 job - 2 machine instance of example 4.1

We have $LB_2(J) = \left\lceil \frac{1}{2}\left(r_6 + r_4 + \sum_{j\in J} p_j + q_5 + q_6\right)\right\rceil = 510$. Both the head and the tail of job $j_0 = 6$ are used in the computation of LB_2. Thus, we have to consider the two following alternatives:

i) job 6 is processed alone on one machine and all other jobs are processed on the other machine: the corresponding lower bound is

$$\max\{r_6 + p_6 + q_6; LB_2^*(J\backslash\{6\}, \emptyset, 1)\} = \max\{123; r_4 + \sum_{j\in J\backslash\{6\}} p_j + q_5\} = 897$$

ii) we cannot use both the head and the tail of job 6: the corresponding lower bound is $LB_2^*(J, \{6\}, 2)$.

The computation of $LB_2^*(J, \{6\}, 2)$ is described as follows. We have $L_r = \{6, 4, 7, 3\}$ and $L_q = \{5, 6, 1, 2\}$. We have $n_{a,b} = 1$ for all $a, b = 2, ..., 4$. Thus, the pairs (a, b) satisfying equation (4.1) are $(2, 3)$ and $(3, 2)$.

For $(a, b) = (2, 3)$, we have $j_r = 4$ and $j_q = 1$. Thus, $R_1 = \{4\}$ and $Q_1 = \{5, 1\}$. Therefore, $R = \{4, 6\}$, $Q = \{5, 1\}$ and $f(R, Q) = 63$.

For $(a, b) = (3, 2)$, we have $j_r = 7$ and $j_q = 6$. Thus, $R_1 = \{4, 7\}$ and $Q_1 = \{5\}$. Therefore, $R = \{4, 7\}$, $Q = \{5, 6\}$ and $f(R, Q) = 69$.

Consequently, we have $R^* = \{4, 6\}$, $Q^* = \{5, 1\}$ and

$$LB_2^*(J, \{6\}, 2) = \left\lceil \frac{1}{2}\left(\sum_{j\in R^*} r_j + \sum_{j\in J} p_j + \sum_{j\in Q^*} q_j\right)\right\rceil = 515$$

Finally, $ALB_2(J) = \min\{897, 515\} = 515$.

4.1.3 Jackson's pseudo preemptive lower bound

Carlier and Pinson (1998) introduced the Jackson's Pseudo Preemptive Schedule ($JPPS$) and showed that it provides a polynomial lower bound for $P|r_j, q_j|C_{\max}$. The $JPPS$ is a generalization of Jackson's preemptive schedule which makespan is a tight lower bound for $1|r_j, q_j|C_{\max}$. In a pseudo preemptive schedule, any operation is allowed to be preempted, and each machine can handle several jobs at one time. Moreover, each job can be processed by more than one machine at one time. Each job j is thus processed, at time t, by a number of machines, denoted by $\alpha_j(t)$, which is not necessarily integer.

Before describing the construction of $JPPS$, some notations have to be introduced. For each job j, is defined at the current time t, its remaining processing time $a_j(t)$ and its complete tail $c_j(t) = q_j + a_j(t)$. A job j is said to be *partially available* at time t if $a_j(t) = p_j - (t - r_j)$, and is said to be *totally available* at time t if $a_j(t) > p_j - (t - r_j)$. In $JPPS$, the set of scheduled jobs between two consecutive decision times is referred to as the schedule block. Two fundamental procedures are required for constructing $JPPS$: the construction of the schedule block at a decision time t, and the computation of the next decision time.

Construction of schedule blocks

In $JPPS$, the jobs with maximal complete tail are scheduled at a maximal rate which depends on their partial or total availability. A subset of jobs with the same priority (complete tail) and the same status (partially or totally available) are scheduled at the same rate. A schedule block is composed of two sets P and T such that jobs of P are scheduled at a rate $\alpha_P = 1$, and those of T are scheduled at a rate $\alpha_T = \dfrac{m - |P|}{|T|}$. The construction of the sets P and T is described below.

Let t be any decision time in $JPPS$, and A the set of available jobs at this time. Let $\xi_1, \xi_2, ..., \xi_k$ be the different complete tails of the elements of A ranked in decreasing order. Define, for $h = 1, 2, ..., k$, the sets P_h and T_h such that :

$$P_h = \{j \in A; c_j(t) = \xi_h \text{ and } a_j(t) = p_j - (t - r_j)\}$$

$$T_h = \{j \in A; c_j(t) = \xi_h \text{ and } a_j(t) > p_j - (t - r_j)\}$$

Denote by s the smallest integer for which one of the following conditions holds :

i) $|P_1| + |P_2| + \cdots + |P_s| \geq m$.

ii) $T_s \neq \emptyset$.

iii) $s = k$.

If $|P_s \cup T_s| > m - |P_1 \cup P_2 \cup ... \cup P_{s-1}|$, then $P = P_1 \cup P_2 \cup ... \cup P_{s-1}$ and $T = P_s \cup T_s$. Otherwise, $P = P_1 \cup P_2 \cup ... \cup P_s$ and $T = T_s$. With no loss of generality, P_0 and T_0 are set equal to the empty set, and $T = \emptyset$ if and only if $\alpha_T = 0$.

Computation of decision times

Let t be any decision time in $JPPS$. The next decision time is the time at which the current schedule block can be modified. This can be done only when at least one of the five following events occurs :

E_1 : A not in-process job becomes available.

E_2 : An in-process job is completed.

E_3 : A not in-process available job enters into the process : the priorities of out of process jobs remain unchanged whereas those of in-process jobs decrease along the time. Thus, the maximal priority of an out of process job can match the priority of an in-process job.

E_4 : An in-process job moves from T to P : if an in-process job $j \in T$ is scheduled, at time t, at a rate $\alpha_T > 1$, then there exists clearly a time $t' > t$ such that $a_j(t') = p_j - (t' - r_j)$.

E_5 : An in-process job moves from P to T : assume that, at time t, jobs of T are scheduled at a rate $\alpha_T < 1$. Since jobs of P are scheduled at a rate $\alpha_P = 1$, their priorities will decrease more quickly than those of T. Therefore, there exists a time where the minimal priority of a job of P matches the priority of a job of T.

Let $c_{\max} = \max_{j \in A \setminus (P \cup T)} c_j$ and denote by t_i the minimal time at which an event E_i can occur $(i = 1, ..., 5)$. The decision time following t is equal to $\min_{1 \leq i \leq 5} t_i$. The computation of t_i $(i = 1, ..., 5)$ is described in the following :

$$t_1 = \begin{cases} +\infty & \text{if } t \geq r_j \text{ for all } j \in J \\ \min\limits_{\substack{j \in J \\ t < r_j}} r_j & \text{otherwise} \end{cases}$$

$$t_2 = \begin{cases} t + \min\limits_{j \in P \cup T} \frac{a_j(t)}{\alpha_j(t)} & \text{if } \alpha_T \neq 0 \\ t + \min\limits_{j \in P} a_j(t) & \text{otherwise} \end{cases}$$

$$t_3 = \begin{cases} t + \min\limits_{j \in P \cup T} \frac{c_j(t) - c_{\max}}{\alpha_j(t)} & \text{if } P \cup T \neq A \text{ and } \alpha_T \neq 0 \\ t + \min\limits_{j \in P} c_j(t) - c_{\max} & \text{if } P \cup T \neq A \text{ and } \alpha_T = 0 \\ +\infty & \text{otherwise} \end{cases}$$

$$t_4 = \begin{cases} t + \frac{1}{\alpha_T - 1} \min_{j \in T}(t + a_j(t) - r_j - p_j) & \text{if } \alpha_T > 1 \\ +\infty & \text{otherwise} \end{cases}$$

$$t_5 = \begin{cases} t + \min_{j \in P} \frac{c_j(t) - c_T}{1 - \alpha_T} & \text{if } P \neq \emptyset \text{ and } 0 < \alpha_T < 1 \\ +\infty & \text{otherwise} \end{cases}$$

Clearly, if at the current decision time, the set of unscheduled jobs is not empty and $A = \emptyset$, then the next decision time is t_1.

Carlier and Pinson (1998) showed that $JPPS$ can be constructed in $O(n \log n + nm \log m)$ time.

Example 4.2: We describe the construction of $JPPS$, depicted in figure 4.1, for the 7 job - 2 machine instance given by Table 4.2. The main operations performed for the construction of the schedule are detailed in Table 4.3. The makespan of $JPPS$ is equal to 36.

j	1	2	3	4	5	6	7
r_j	2	22	13	1	1	14	13
p_j	9	6	9	10	2	8	5
q_j	21	7	12	23	29	11	15

Table 4.2. Data of the 7 job - 2 machine instance of example 4.2

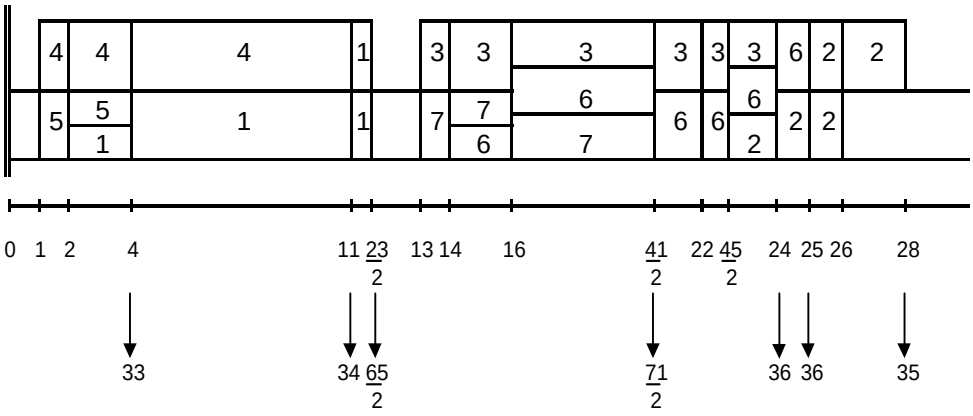


Figure 4.1. Example of $JPPS$

t	P	T	α_T
$t_1 = 1$	$\{4, 5\}$	$\emptyset$	0
$t_1 = 2$	$\{4\}$	$\{5, 1\}$	$\frac{1}{2}$
$t_2 = 4$	$\{4\}$	$\{1\}$	1
$t_2 = 11$	$\emptyset$	$\{1\}$	2
$t_2 = \frac{23}{2}$	$\emptyset$	$\emptyset$	0
$t_1 = 13$	$\{3, 7\}$	$\emptyset$	0
$t_1 = 14$	$\{3\}$	$\{7, 6\}$	$\frac{1}{2}$
$t_5 = 16$	$\emptyset$	$\{3, 6, 7\}$	$\frac{2}{3}$
$t_2 = \frac{41}{2}$	$\emptyset$	$\{3, 6\}$	1
$t_1 = 22$	$\emptyset$	$\{3, 6\}$	1
$t_3 = \frac{45}{2}$	$\emptyset$	$\{3, 6, 2\}$	$\frac{2}{3}$
$t_2 = 24$	$\emptyset$	$\{6, 2\}$	1
$t_2 = 25$	$\emptyset$	$\{2\}$	2
$t_4 = 26$	$\{2\}$	$\emptyset$	1

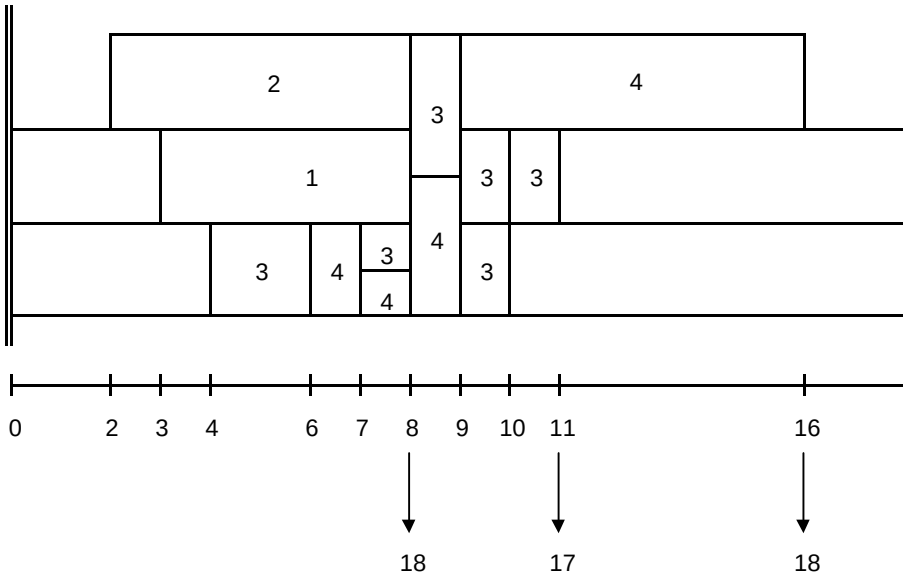
Table 4.3. Operations for $JPPS$ construction

In the sequel, the value $\lceil C_{\max}(JPPS) \rceil$ will be referred as the Jackson's Pseudo Preemptive Bound ($JPPB$).

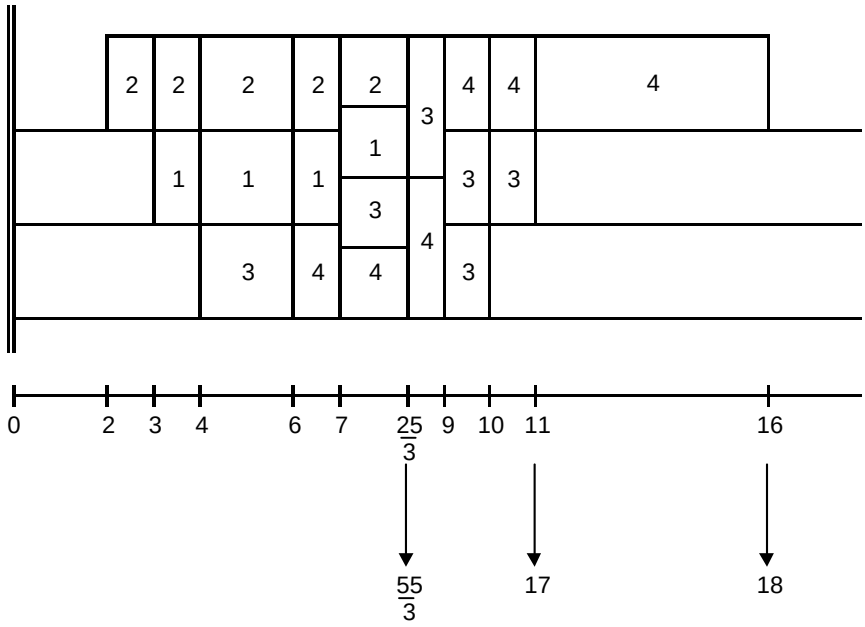
It is noteworthy that, contrarily to the fact that Jackson's preemptive schedule is optimal for the one machine preemptive problem, the $JPPS$ does not provide the optimal solution of the m machine pseudo preemptive problem. Indeed, figure 4.2a shows a pseudo preemptive schedule, which makespan is less than $JPPS$ one (see figure 4.2b), for the 4 job - 3 machine instance given by Table 4.4.

j	1	2	3	4
r_j	3	2	4	6
p_j	5	6	7	10
q_j	10	10	6	2

Table 4.4. Data of a 4 job - 3 machine instance



4.2a. A pseudo preemptive schedule



4.2b. $JPPS$ schedule

Figure 4.2. $JPPS$ is not optimal

4.1.4 The preemptive lower bound

If we relax the nonpreemption constraint, then we obtain a preemptive parallel machine problem denoted by $P|r_j, q_j, pmtn|C_{\max}$. Obviously, the makespan of an optimal preemptive schedule, denoted hereafter by the *Preemptive Lower Bound (PLB),* constitutes a strong lower bound for $P|r_j, q_j|C_{\max}$.

The optimal makespan of a $P|r_j, q_j, pmtn|C_{\max}$ instance is obtained after repeatedly checking the existence of a preemptive schedule with $C_{\max}$ equal to an integer trial value C. If C^- and C^+ denote a lower and upper bound on the trial value C, then the optimal $C_{\max}$ is computed using a bisection search on the interval $[C^-, C^+]$.

Assume that there exists a preemptive schedule with $C_{\max} = C$. That is, each job $j \in J$ has to finish processing before $C - q_j$. This can be assured by setting a deadline $d_j = C - q_j$ for all $j \in J$ (i.e. job j is constrained to be completed before d_j). Horn (1974) proposed a polynomial algorithm to test the existence of a preemptive schedule subject to release dates and deadlines. This approach consists in transforming the feasibility problem into an equivalent maximum flow one. Horn's procedure is described in the following.

Let $\{e_1, ..., e_H\}$ be the set containing all release dates and deadlines of all jobs ranked in increasing order. A time interval $E_h = [e_h, e_{h+1}]$ is defined for each $h = 1, ..., H-1$. Consider the flow network composed of job nodes $\{J_1, ..., J_n\}$, interval nodes $\{E_1, ..., E_{H-1}\}$, a source node s, and a sink node t. For each job node J_j $(j = 1, ..., n)$, there is an arc (s, J_j) with capacity p_j. For each interval node E_h $(h = 1, ..., H-1)$, there is an arc (E_h, t) with capacity $m(e_{h+1} - e_h)$. There is an arc (J_j, E_h) with capacity $e_{h+1} - e_h$ if and only if $r_j \leq e_h$ and $e_{h+1} \leq d_j$. A preemptive schedule with $C_{\max}$ equal to C is defined as an assignment of portions of the processing time of each job j $(j = 1, ..., n)$ to different time intervals E_h $(h = 1, ..., H-1)$. Such a schedule is feasible if and only if the maximum flow value is equal to $\sum_{j=1}^{n} p_j$. Indeed, since the total capacity of arcs leaving s amounts to $\sum_{j=1}^{n} p_j$, then the maximum flow cannot exceed $\sum_{j=1}^{n} p_j$. On the other hand, the value of the total flow from all job nodes to all interval nodes in a maximum flow

cannot be less than $\sum_{j=1}^{n} p_j$. This is due to the fact that, for all $j \in J$, we have $r_j + p_j \leq d_j$. Moreover, in a maximum flow, the value of the total flow from the interval nodes to t cannot be less than the value of the total flow from the job nodes to the interval nodes. Otherwise, there are more jobs that have to be processed in the same interval than there are available machines.

The computation of the maximum flow requires $O(N^3)$ time, where N is the number of nodes in the network. We have a maximum of $3n + 1$ nodes. Thus, checking the existence of a preemptive schedule with $C_{\max}$ equal to C requires $O(n^3)$ time. By performing a bisection search on the interval $[C^-, C^+]$, the lower bound PLB can be computed in $O(n^3(\log n + \log p_{\max}))$ time (Labetoulle et al., 1984).

It is worth noting that $JPPB \leq PLB$. Thus, $JPPB$ can be useful for the computation of PLB by setting $C^- = JPPB$. The upper bound C^+ can be computed using Jackson's algorithm (cf. page 84).

Example 4.3: Consider the 3 job-2 machine instance defined by Table 4.5.

j	1	2	3
r_j	23	0	0
p_j	14	45	74
q_j	52	43	0

Table 4.5. Data of the 3 job-2 machine instance of example 4.3

We have $C^- = JPPB = 89$ and $C^+ = 111$ (Figure 4.3).

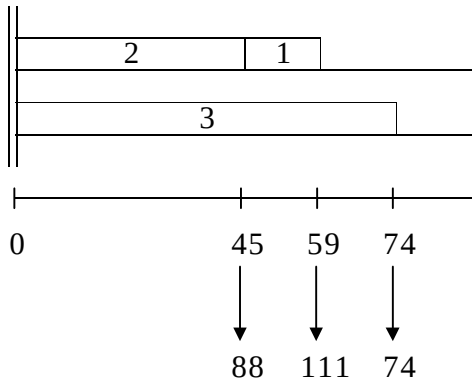


Figure 4.3. A feasible schedule with $C_{\max} = 111$

Assume that there exists a preemptive schedule with $C_{\max}$ equal to the trial value $C = 89$. Thus, the deadlines are set to $d_1 = 37$, $d_2 = 46$ and $d_3 = 89$. Figure 4.4 depicts the flow network corresponding to this feasibility problem. The maximum flow value is equal to $\sum_{j \in J} p_j = 133$. Therefore, $PLB = 89$. An optimal preemptive schedule with $C_{\max} = 89$ is depicted in Figure 4.5.

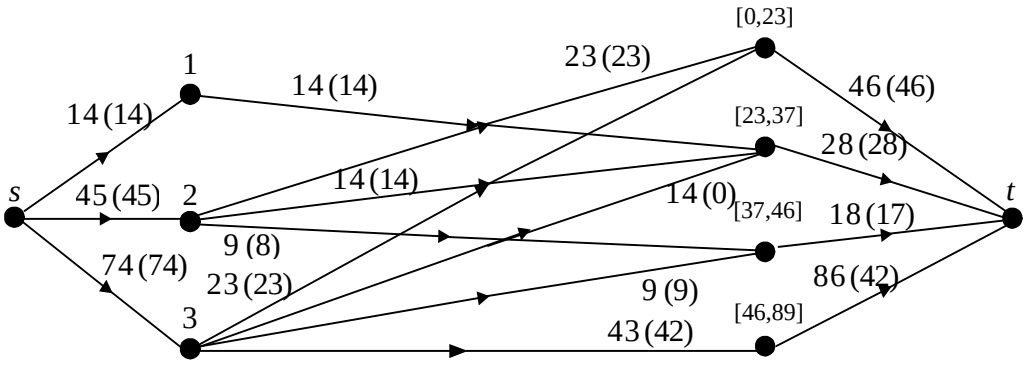


Figure 4.4. The network corresponding to $C = 89$

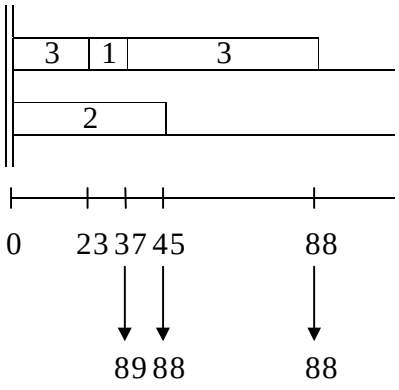


Figure 4.5. An optimal preemptive schedule

4.2 The semi-preemptive lower bound

In this section, we develop a new lower bound which dominates the preemptive bound while having the same complexity. The proposed bound is similar in spirit to PLB in the sense that it consists in checking the feasibility of a schedule with $C_{\max}$ equal to a trial value C. This new lower bound has been published in Haouari and Gharbi (2003).

Firstly, by remarking that the latest starting time of any job $j \in J$ is $d_j - p_j$, and its earliest finishing time is $r_j + p_j$, we can state the following observation:

Observation 4.2

Assume that there exists a job j such that $d_j - r_j < 2p_j$. Then, in any nonpreemptive schedule, there is necessarily one machine which must process job j during the interval $[d_j - p_j, r_j + p_j]$.

According to the above observation, each job j satisfying $d_j - r_j < 2p_j$ is composed of a *fixed* and a *free* part. Its fixed part is the amount of time $2p_j - (d_j - r_j)$ which must be processed in $[d_j - p_j, r_j + p_j]$, and its free part is the amount of time $p'_j = d_j - (r_j + p_j)$ which has to be processed in $[r_j, d_j - p_j] \cup [r_j + p_j, d_j]$. The other jobs are composed only of a free processing part $p'_j = p_j$ which has to be processed in $[r_j, d_j]$. That is, a free part of any job $j \in J$ is $p'_j = \min\{p_j, d_j - (r_j + p_j)\}$.

Clearly, a relaxation of the $P|r_j, q_j|C_{\max}$ can be obtained if the preemption is allowed for only the free parts of the jobs in J. We define a *semi-preemptive* schedule as a schedule where the fixed parts of the jobs are constrained to start and to finish at fixed times with no preemption, whereas the free parts can be preempted.

The feasibility of a semi-preemptive schedule with $C_{\max}$ equal to a trial value C can be checked as follows: First, note that Observation 4.2 provides a simple way to compute a lower bound on the number of machines which are necessarily loaded at any time. Therefore, the following result clearly holds:

Observation 4.3

If there is a time $t \in [0, \max_{j \in J} d_j]$ such that the number of machines loaded at t is strictly greater than m, then there is no feasible semi-preemptive schedule.

Let $S = \{j \in J;\ d_j - r_j < 2p_j\}$ denote the set of jobs having a fixed processing part. Let $e_1, e_2, ..., e_K$ be the different values of r_j $(j \in J)$, d_j $(j \in J)$, $d_j - p_j$ $(j \in S)$ and $r_j + p_j$ $(j \in S)$ ranked in increasing order. We denote by m_k the number of machines which are idle during the time interval $E_k = [e_k, e_{k+1}]$ $(1 \leq k \leq K-1)$ according to Observation 4.2. The feasibility problem can be solved using the following extension of Horn's approach:

Consider the flow network composed of job nodes $\{J_1, ..., J_n\}$, interval nodes $\{E_1, ..., E_{K-1}\}$, a source node s, and a sink node t. For each job node J_j $(j = 1, ..., n)$ such that $p'_j > 0$, there is an arc (s, J_j) with capacity p'_j representing the free part of job j. For each $k = 1, ..., K-1$, there is an arc (E_k, t) with capacity $m_k(e_{k+1} - e_k)$. There is an arc (J_j, E_k) with capacity $e_{k+1} - e_k$ if and only if one of the three following conditions is satisfied:

- $d_j - r_j < 2p_j,\ r_j \leq e_k$ and $e_{k+1} \leq d_j - p_j$
- $d_j - r_j < 2p_j,\ r_j + p_j \leq e_k$ and $e_{k+1} \leq d_j$
- $d_j - r_j \geq 2p_j,\ r_j \leq e_k$ and $e_{k+1} \leq d_j$

Obviously, an interval node E_k with $m_k = 0$ or which is not connected with any job node is dropped from the network. Clearly, a semi-preemptive schedule with $C_{\max}$ equal to C exists if and only if the maximum flow is equal to $\sum_{j \in J} p'_j$.

We have a maximum of $5n+1$ nodes in the corresponding network. Therefore, the feasibility test requires $O(n^3)$ time. The semi-preemptive lower bound, denoted by $SPLB$, is defined as the optimal semi-preemptive $C_{\max}$ obtained after performing a bisection search on the interval $[C^-, C^+]$. The bound $SPLB$ can be computed in $O(n^3(\log n + \log p_{\max}))$.

Clearly, we have:

$$PLB \leq SPLB$$

Example 4.3 (continued)**:** Assume that there exists a semi-preemptive schedule with $C_{\max}$ equal to the trial value $C = 89$. Thus, the deadlines are set to $d_1 = 37$, $d_2 = 46$ and $d_3 = 89$. According to Observation 4.2, there is necessarily one machine which must process job 1 in the interval $[23, 37]$ for 14 units of time. Also, a second machine must process job 2 in

the interval $[1, 45]$ for 44 units of time, and a third machine must process job 3 in the interval $[15, 74]$ for 59 units of time. That is, there must be *three* loaded machines in the interval $[23, 37]$. Therefore, the instance is infeasible (cf. Observation 4.3).

Consider the trial value $C = 103$. The deadlines are set to $d_1 = 51$, $d_2 = 60$ and $d_3 = 103$. According to Observation 4.2, there is necessarily one machine which must process job 2 in the interval $[15, 45]$ for 30 units of time, and a second machine which must process job 3 in the interval $[29, 74]$ for 45 units of time. Thus, the free processing parts are $p'_1 = 14$, $p'_2 = 15$ and $p'_3 = 29$. Figure 4.6 depicts the flow network corresponding to this feasibility problem. The maximum flow value is equal to 56 whereas $\sum_{j \in J} p'_j = 58$. Therefore, the instance is infeasible.

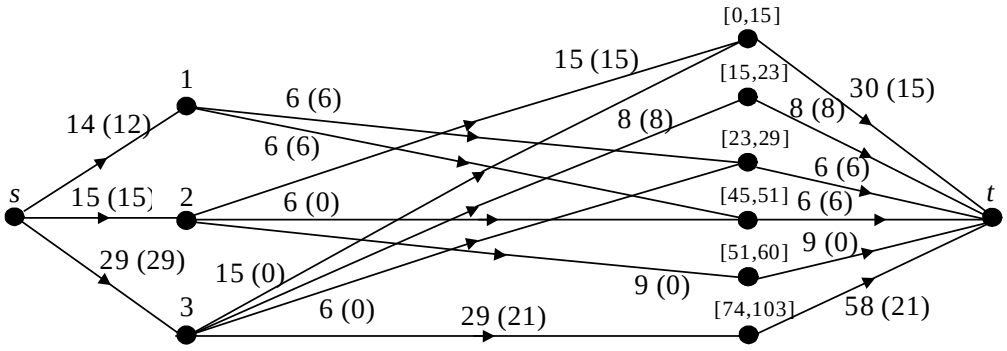


Figure 4.6. The network corresonding to $C = 103$

Now, assume that $C = 104$. The deadlines are set to $d_1 = 52$, $d_2 = 61$ and $d_3 = 104$. According to Observation 4.2, there is necessarily one machine which must process job 2 in the interval $[16, 45]$ for 29 units of time, and a second machine which must process job 3 in the interval $[30, 74]$ for 44 units of time. Thus, the free processing parts are $p'_1 = 14$, $p'_2 = 16$ and $p'_3 = 30$. Figure 4.7 depicts the flow network corresponding to this feasibility problem. The maximum flow value is equal to $\sum_{j \in J} p'_j = 60$. Therefore, $SPLB = 104$. An optimal semi-preemptive schedule with $C_{\max} = 104$ is depicted in Figure 4.8.

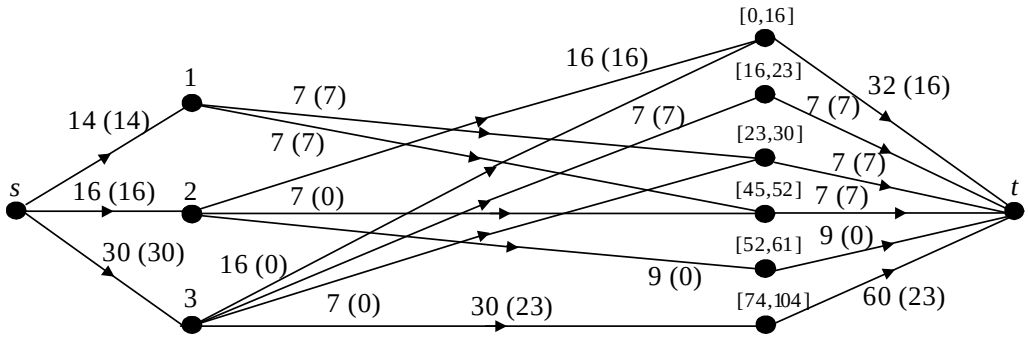


Figure 4.7. The network corresponding to $C = 104$

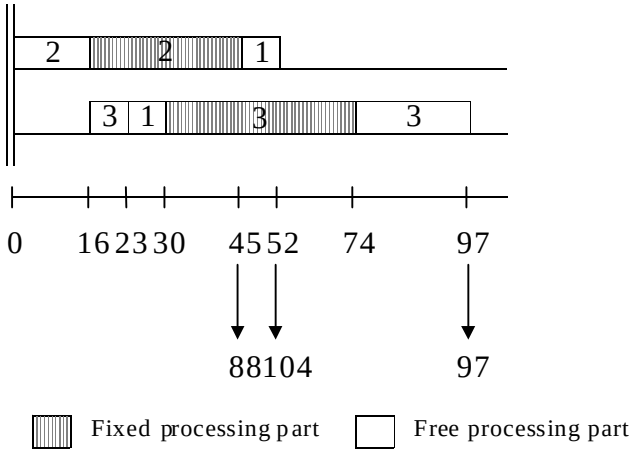


Figure 4.8. An optimal semi-preemptive schedule

4.3 A machine availability based lower bound

In this section, we introduce a bounding scheme based on a new lower bound referred to hereafter as the *Modified General Bound* (MGB). This bound is derived from the so-called *General Bound* (GB) proposed by Webster (1996) for the parallel machine scheduling problem with availability constraints denoted by $P, NC_{inc}||C_{\max}$. In this case, a machine m_i $(i = 1, ..., m)$ is available from time $u_i \geq 0$ onwards.

A relaxation of $P|r_j, q_j|C_{\max}$ as a $P, NC_{inc}||C_{\max}$ may be obtained as follows. Assume that all release dates and delivery times are set equal to zero, and that each machine m_i has an available time $u_i = \overline{r}_i$ $(i = 1, ..., m)$. In the remainder of this section, it is assumed without loss of generality that

the jobs are ranked in non-increasing order of their processing times, and the machines are ranked in nondecreasing order of their availability times.

4.3.1 The general bound

First we present the General Bound (GB) which is based on the two following theorems stated by Webster (1996) for the $P, NC_{inc}||C_{\max}$ problem. The first theorem gives sufficient conditions for the optimality of the LPT rule which schedules the job with the longest processing time on the first available machine. The second gives a sufficient condition for checking the optimality of any non-delay schedule (i.e. a schedule where no machine is kept idle while a job is waiting).

Theorem 4.1

If one of the four following conditions is satisfied, then the LPT rule is optimal.

$\mathbf{C}_1 : p_j/p_{j+1}$ *is integer for* $j = 1, ..., n-1$.
$\mathbf{C}_2 : n \leq m$ *and* $u_n - u_1 \leq p_j$ *for* $j = 1, ..., n-1$.
$\mathbf{C}_3 : u_i = 0$ *for all* $i = 1, ..., m$, *and* $p_{m-1} \geq p_m + \cdots + p_{n-2}$.
$\mathbf{C}_4 : u_i = 0$ *for all* $i = 1, ..., m$; $n \leq 2m$ *and* $p_{2m-n+1} \leq 2p_n$.

Theorem 4.2

Let $t_1, ..., t_n$ *be the job starting times in a nondelay schedule* σ *for a* $P, NC_{inc}||C_{\max}$, *and let* j_0 *be the last job to finish processing. If* p_j/p_{j_0} *is integer for all* j *such that* $t_j < t_{j_0}$, *then* σ *is optimal.*

Consider two positive integers n' and ρ, and let τ be a common divisor of $\rho, (p_1 - \left\lfloor \frac{p_1}{\rho} \right\rfloor \rho), ..., (p_n - \left\lfloor \frac{p_n}{\rho} \right\rfloor \rho)$. A relaxed problem of $P, NC_{inc}||C_{\max}$ can be obtained if the set J of n jobs is replaced by $J'_\rho = J_1 \cup J_2 \cup J_3$ such that :

- J_1 is a set of n' ($n' \leq n$) jobs with respective processing times $\left\lfloor \frac{p_1}{\rho} \right\rfloor \rho$, $\left\lfloor \frac{p_2}{\rho} \right\rfloor \rho, ..., \left\lfloor \frac{p_{n'}}{\rho} \right\rfloor \rho$.
- J_2 is a set of $\left\lfloor \frac{p_{n'+1}}{\rho} \right\rfloor + \cdots + \left\lfloor \frac{p_n}{\rho} \right\rfloor$ jobs with equal processing times ρ.
- J_3 is a set of $(p_1 - \left\lfloor \frac{p_1}{\rho} \right\rfloor \rho)/\tau + \cdots + (p_n - \left\lfloor \frac{p_n}{\rho} \right\rfloor \rho)/\tau$ jobs with equal processing times τ.

Denote by $P(J'_\rho)$ the $P, NC_{inc}||C_{\max}$ defined on J'_ρ. The optimal makespan of $P(J'_\rho)$ is defined as a general lower bound on the optimal one of $P(J)$. It will be denoted by $GB_\rho(J)$.

The number n' is selected such that $P(J_1)$, can be optimally solved using LPT rule. Consequently, the problem $P(J'_\rho)$, can also be optimally solved using LPT rule. In fact, consider the schedule σ obtained using LPT rule on the set J'_ρ, and denote by j_0 the last job to finish processing in σ. Two cases are possible :

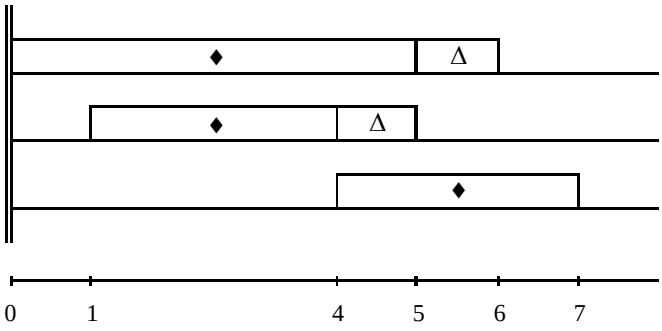
i) The job j_0 is in J_1 : In this case σ is a feasible schedule of $P(J'_\rho)$ with makespan equal to the one of the LPT schedule of $P(J_1)$. Since $P(J_1)$ is a relaxation of $P(J'_\rho)$, and the LPT schedule is optimal for $P(J_1)$, then σ is optimal.

ii) The job j_0 is in $J_2 \cup J_3$: Note that, in an LPT schedule, jobs of J_1 are scheduled first, then those of J_2, and finally those of J_3. That is, each job j starting before j_0 in σ, is such that p_j/p_{j_0} is integer. Therefore, by theorem 4.2, we conclude that σ is optimal.

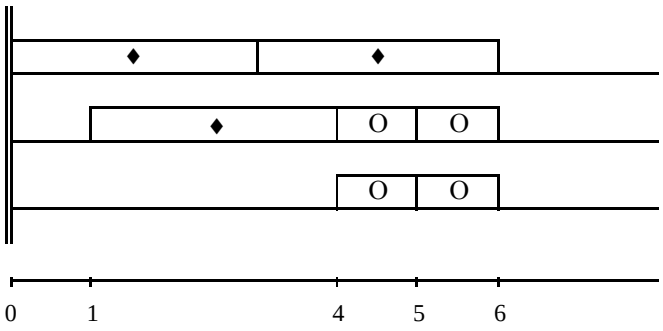
Clearly, $GB_\rho(J)$ is maximized if n' and τ are selected to be as large as possible for a given value of ρ. Consequently, for a given ρ, the number n' is selected to be the largest number for which at least one of the four conditions given by theorem 4.1 is satisfied according to the data of the set J_1. The number τ is set to $\gcd(\rho, (p_1 - \left\lfloor \frac{p_1}{\rho} \right\rfloor \rho), ..., (p_n - \left\lfloor \frac{p_n}{\rho} \right\rfloor \rho))$.

Webster (1996) proposed two values of ρ for which $P(J'_\rho)$ is a very tight relaxation of $P(J)$. These values are $\rho_1 = \gcd(p_1, p_2, ..., p_n)$ and $\rho_2 = p_{j_0}$, where j_0 is the last job to finish processing in the LPT schedule constructed for J. Example 4.4 shows that there is no dominance relation between $GB_{\rho_1}(J)$ and $GB_{\rho_2}(J)$. We will refer to the general bound $GB(J)$ as the maximum value between $GB_{\rho_1}(J)$ and $GB_{\rho_2}(J)$. The bound GB can be computed in $O(n \log n)$ time.

Example 4.4: Consider the 5 job - 3 machine instance of $P, NC_{inc}||C_{\max}$ with machine availabilities $\{0, 1, 4\}$ and processing times $\{5, 3, 3, 1, 1\}$. For $\rho_1 = 1$, we have $n' = 3$, $|J_2| = 2$, $J_3 = \emptyset$ and $GB_{\rho_1}(J) = 7$ (see figure 4.9a). For $\rho_2 = 3$, we have $n' = 3, J_2 = \emptyset$, $\tau = 1, |J_3| = 4$ and $GB_{\rho_2}(J) = 6$ (see figure 4.9b).



4.9a. $GB_{\rho_1}(J) = 7$

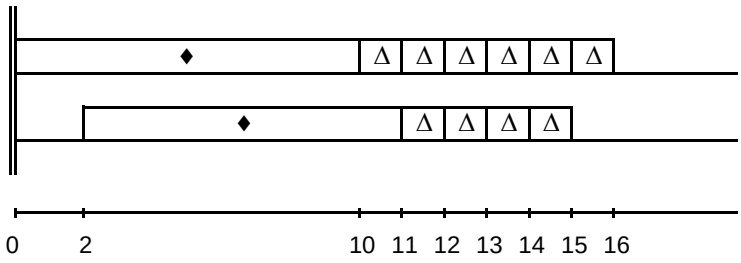


4.9b. $GB_{\rho_2}(J) = 6$

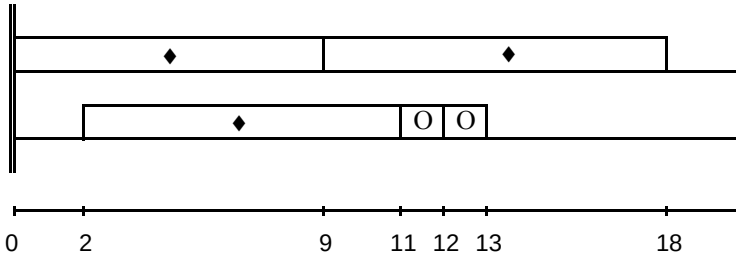
♦ *Jobs of* J_1 $\quad$ Δ *Jobs of* J_2 $\quad$ O *Jobs of* J_3

Figure 4.9. $GB_{\rho_1}(J) > GB_{\rho_2}(J)$

A case where $GB_{\rho_1}(J) < GB_{\rho_2}(J)$ is the following 4 job - 2 machine instance which machine availabilities are $\{0, 2\}$ and processing times are $\{10, 9, 9, 1\}$. For $\rho_1 = 1$, we have $n' = 2$, $|J_2| = 10$, $J_3 = \emptyset$ and $GB_{\rho_1}(J) = 16$ (see figure 4.10a). For $\rho_2 = 9$, we have $n' = 3$, $J_2 = \emptyset$, $\tau = 1, |J_3| = 2$ and $GB_{\rho_2}(J) = 18$ (see figure 4.10b).



4.10a. $GB_{\rho_1}(J) = 16$



4.10b. $GB_{\rho_2}(J) = 18$

♦ *Jobs of* J_1 Δ *Jobs of* J_2 O *Jobs of* J_3

Figure 4.10. $GB_{\rho_1}(J) < GB_{\rho_2}(J)$

4.3.2 The modified general bound

In order to introduce other valid lower bounds, we present two lemmas which are consequences of the application of the preprocessing algorithm (cf. page 27). These lemmas show that adding some specific sets of dummy jobs has no effect on the optimal makespan.

Lemma 4.1

Consider the sets $D_1(J)$ and $D_2(J)$, each includes m dummy jobs such that :

$$D_1(J) = \{j_k \ (k = 1, ..., m) \ / \ r_{j_k} = 0, \ p_{j_k} = \overline{r}_k(J) and \ q_{j_k} = \overline{q}_m(J)\}$$

$$D_2(J) = \{j_k \ (k = 1, ..., m) \ / \ r_{j_k} = \overline{r}_m(J), \ p_{j_k} = \overline{q}_k(J) and \ q_{j_k} = 0\}$$

Then,

$$C^*_{\max}(J) = C^*_{\max}(J \cup D_1(J) \cup D_2(J))$$

Proof. Clearly, jobs of $D_1(J)$ and $D_2(J)$ satisfy the conditions (3.1) and (3.2) (cf. page 28), respectively. Therefore,

$$C^*_{\max}(J \cup D_1(J) \cup D_2(J)) = \max\{C^*_{\max}(J), LB_0(D_1(J)), LB_0(D_2(J))\}$$

Denote by $\overline{J}$ the set obtained after applying preprocessing algorithm to J, and by $D_1(\overline{J})$ the set of m dummy jobs such that :

$$D_1(\overline{J}) = \left\{j_k \ (k = 1, ..., m) \ / \ r_{j_k} = 0, \ p_{j_k} = \overline{r}_k(\overline{J}) \text{ and } q_{j_k} = \overline{q}_m(\overline{J})\right\}$$

Note that $\overline{r}_k(\overline{J}) < \min\limits_{j \in \overline{J}} (r_j + p_j)$ for all $k = 1, ..., m$. Indeed, if there was a job $j_0 \in \overline{J}$ such that $r_{j_0} + p_{j_0} = \min\limits_{j \in \overline{J}} (r_j + p_j)$ and satisfying $r_{j_0} + p_{j_0} \leq \overline{r}_k(\overline{J})$, then j_0 would be in J_R and not in $\overline{J}$.

Let j'_0 denote a job of $\overline{J}$ with tail equal to $\overline{q}_m(\overline{J})$. Thus, for every $j_k \in D_1(\overline{J})$ $(k = 1, ..., m)$, we have :

$$\begin{aligned} r_{j_k} + p_{j_k} + q_{j_k} &= \overline{r}_k(\overline{J}) + \overline{q}_m(\overline{J}) \\ &< \min_{j \in \overline{J}} (r_j + p_j) + \overline{q}_m(\overline{J}) \\ &\leq r_{j'_0} + p_{j'_0} + q_{j'_0} \\ &\leq LB_0(\overline{J}) \\ &\leq C^*_{\max}(J) \end{aligned}$$

Therefore, $LB_0(D_1(\overline{J})) < C^*_{\max}(J)$.

On the other hand, we have $LB_0(D_1(J)) \leq LB_0(D_1(\overline{J}))$. So, $LB_0(D_1(J)) < C^*_{\max}(J)$. Similarly, it can be proven that $LB_0(D_2(J)) < C^*_{\max}(J)$. ■

Lemma 4.2

Consider the sets $D'_1(J)$ and $D'_2(J)$, each includes $m-1$ dummy jobs such that :

$$D'_1(J) = \{j_k(1 \leq k \leq m-1)/\ r_{j_k} = \overline{r}_1(J), p_{j_k} = \overline{r}_{k+1}(J) - \overline{r}_1(J), q_{j_k} = \overline{q}_m(J)\}$$

$$D'_2(J) = \left\{j_k(1 \leq k \leq m-1)/\ r_{j_k} = \overline{r}_m(J), p_{j_k} = \overline{q}_{k+1}(J) - \overline{q}_1(J), q_{j_k} = \overline{q}_1(J)\right\}$$

Then,

$$C^*_{\max}(J) = C^*_{\max}(J \cup D'_1(J) \cup D'_2(J))$$

Proof. The proof of Lemma 4.2 is similar to that of Lemma 4.1 and is therefore omitted. ■

As described above, the $P, NC_{inc}||C_{\max}$ can be formulated as a relaxation of $P|r_j, q_j|C_{\max}$. Therefore, the bound GB is a valid lower bound for $P|r_j, q_j|C_{\max}$. Note that the relaxation scheme takes into consideration the m smallest heads of the jobs. Unfortunately, their tails are totally ignored. An attempt to consider the m smallest tails is to add to the set J, the set $D_2(J)$ of dummy jobs described in Lemma 4.1. Since the obtained $P|r_j, q_j|C_{\max}$ problem has the same optimal makespan as the initial one, then the bound $GB(J \cup D_2(J))$ remains a lower bound for the $P|r_j, q_j|C_{\max}$ problem defined for J.

Another valid lower bound can be derived if the aforementioned set of dummy jobs $D'_2(J)$ is added to the set J. Note that, in this case, each machine has to wait at least an amount of time equal to $\overline{q}_1(J)$ after finishing processing. Consequently, $GB(J \cup D'_2(J)) + \overline{q}_1(J)$ is a valid lower bound for the $P|r_j, q_j|C_{\max}$ problem defined for J.

It is worth noting that when the Backward problem is considered, then two additional lower bounds $GB^{-1}(J \cup D_1(J))$ and $GB^{-1}(J \cup D'_1(J)) + \overline{r}_1(J)$ can also be derived, where GB^{-1} is the general bound computed for the relaxation of the Backward problem. The following proposition shows that all the four derived general bounds dominate LB_2.

Proposition 4.2 :

i) $GB(\overline{J} \cup D_2(\overline{J})) \geq LB_2(\overline{J})$

ii) $GB(\overline{J} \cup D_2'(\overline{J})) + \overline{q}_1(\overline{J}) \geq LB_2(\overline{J})$

iii) $GB^{-1}(\overline{J} \cup D_1(\overline{J})) \geq LB_2(\overline{J})$

iv) $GB^{-1}(\overline{J} \cup D_1'(\overline{J})) + \overline{r}_1(\overline{J}) \geq LB_2(\overline{J})$

Proof. The proof of (ii) is similar to that of (i). The proofs of (iii) and (iv) can be derived by symmetry from those of (i) and (ii), respectively. Thus, only (i) will be proven.

Clearly, for any set S, we have :

$$\left\lceil \frac{1}{m} \left(\overline{r}_1(S) + \overline{r}_2(S) + \cdots + \overline{r}_m(S) + \sum_{j \in S} p_j \right) \right\rceil \leq GB(S)$$

If the latter inequality is applied to the set $\overline{J} \cup D_2(\overline{J})$, then we obtain :

$$\left\lceil \frac{1}{m} \left(\sum_{k=1}^{m} \overline{r}_k(\overline{J}) + \sum_{j \in \overline{J}} p_j + \sum_{k=1}^{m} \overline{q}_k(\overline{J}) \right) \right\rceil \leq GB(\overline{J} \cup D_2(\overline{J}))$$

That is, $LB_2(\overline{J}) \leq GB(\overline{J} \cup D_2(\overline{J}))$. ■

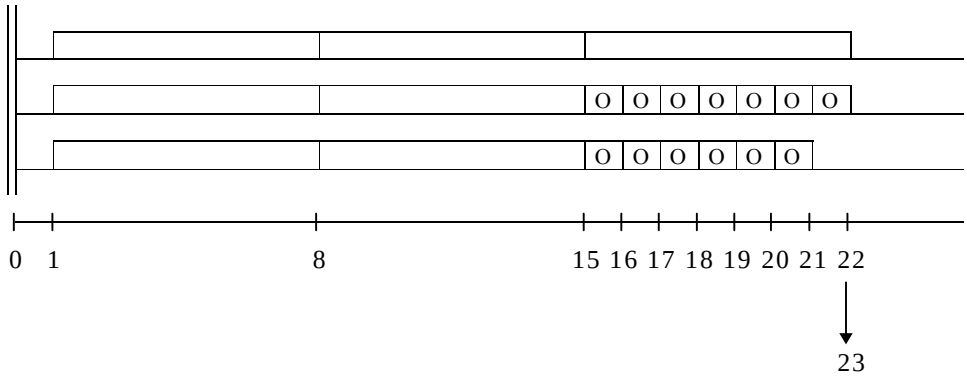
Note that there is no dominance relation between the four general bounds mentioned above. Our experimental results show that these bounds are equal in nearly all cases with a slight advantage for the general bound $GB(\overline{J} \cup D_2'(\overline{J})) + \overline{q}_1(\overline{J})$. This bound will be referred to as the *Modified General Bound,* and will be denoted by $LB_3(J)$. Clearly, the computation of LB_3 requires $O(n \log n)$ time.

Example 4.5: Consider the 7 job - 3 machine instance defined by Table 4.6.

j	1	2	3	4	5	6	7
r_j	1	1	1	2	2	2	8
p_j	9	8	4	8	7	1	9
q_j	8	8	9	8	8	10	1

Table 4.6. Data of the 7 job - 3 machine instance of example 4.5

We have $\bar{J} = J$ and $D_2'(\bar{J}) = \{j_1, j_2\}$ such that $p_{j_1} = p_{j_2} = 7$. The $P, NC_{inc}||C_{\max}$ problem defined on $\bar{J} \cup D_2'(\bar{J})$ is such that the machine availabilities are $\{1, 1, 1\}$. For $\rho_2 = 7$, we have $n' = 7$, $J_2 = \emptyset$, $\tau = 1$, $|J_3| = 11$ and $LB_3(J) = 23$ (see Figure 4.11).



O *Jobs of* J_3

Figure 4.11. $LB_3(J) = 23$

It is noteworthy that two additional variants of $LB_3(J)$ can be constructed :

i) By considering the set J instead of $\overline{J}$.

ii) By computing the general bound without adding dummy jobs.

The resulting bound does not dominate the bound $LB_2(J)$. Moreover, our experimental results showed that $LB_3(J)$ consistently dominates its variants.

4.4 A bin-packing based lower bound

In this section, the value $\bar{p}_j(J)$ denotes the j^{th} smallest processing time in J. A simple way to relax the $P|r_j, q_j|C_{\max}$ is to assume that all the heads and tails are equal to $\min_{j\in J} r_j$ and $\min_{j\in J} q_j$, respectively. Let $C^*_{\max}(J)$ denote the optimal makespan of the $P|r_j, q_j|C_{\max}$ defined on J. The following result clearly holds:

$$\min_{j\in J} r_j + LB_{P||C_{\max}}(J) + \min_{j\in J} q_j \leq C^*_{\max}(J)$$

where $LB_{P||C_{\max}}(J)$ denotes a lower bound for the $P||C_{\max}$ defined on J. The $P||C_{\max}$ is a fundamental scheduling problem that has been extensively investigated by many researchers, and several lower bounds have been proposed in the literature (Dell'Amico and Martello, 1995). These bounds include $\max_{j\in J} p_j$, $\left\lceil \frac{1}{m}\sum_{j\in J} p_j \right\rceil$, and $\bar{p}_{n-m}(J) + \bar{p}_{n-m+1}(J)$ to quote just a

few. Therefore, any of these bounds can be used to derive a lower bound for $P|r_j, q_j|C_{\max}$. For instance, taking $LB_{P||C_{\max}}(J) = \left\lceil \frac{1}{m} \sum_{j \in J} p_j \right\rceil$ yields $LB_1(J)$ (cf. page 34). However, a much more interesting lower bound can be derived from the optimal solution of a Bin Packing Problem (BPP). The BPP is an $\mathcal{NP}$-hard optimization problem which can be described as follows: Given a set of weighted items and a set of bins, each item has to be assigned to one bin so that the total weight of the items in each bin does not exceed a given capacity C. The objective is to minimize the number of used bins (Martello and Toth, 1990). The $P||C_{\max}$ is related to the BPP in the following way. Given a trial value C of the optimal makespan, consider the decision problem which consists in checking whether the n jobs can be processed on the m machines such that the maximum completion time does not exceed C. A positive answer to this feasibility problem is provided if the minimal number of bins (machines) does not exceed m in the BPP defined for the n items (jobs) with weights p_j and bins' capacity C. Otherwise, we conclude that a valid lower bound for the $P||C_{\max}$ is $C + 1$.

Dell'Amico and Martello (1995) proposed a strong BPP based lower bound for the $P||C_{\max}$ which can be described as follows: For each integer $p = \bar{p}_1(J), \bar{p}_2(J), ..., \min\left\{\bar{p}_{n-m-1}(J), \frac{C}{2}\right\}$, the subsets J_1, J_2 and J_3 are defined by:

$$J_1 = \{j \in J;\ C - p < p_j\}$$
$$J_2 = \left\{j \in J;\ \frac{C}{2} < p_j \leq C - p\right\}$$
$$J_3 = \left\{j \in J;\ p \leq p_j \leq \frac{C}{2}\right\}$$

The two proposed lower bounds of the minimal number of used bins are:

$$BPP_1(C,p) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in J_3} p_j - C\,|J_2| + \sum_{j \in J_2} p_j}{C} \right\rceil\right)$$

$$BPP_2(C,p) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{|J_3| - \sum_{j \in J_2} \left\lfloor \frac{C - p_j}{p} \right\rfloor}{\left\lfloor \frac{C}{p} \right\rfloor} \right\rceil\right)$$

Clearly, if $\max_p \{\max(BPP_1(C,p), BPP_2(C,p))\} > m$ then $C+1$ is a valid lower bound for the $P||C_{\max}$. Therefore, an enhanced lower bound for the $P||C_{\max}$ can be computed through a bisection search on the interval $[L,U]$, where L and U denote a lower and upper bound on the optimal makespan, respectively. The lower bound is:

$$L = \max\left\{\max_{j\in J} p_j, \left\lceil \frac{1}{m}\sum_{j\in J} p_j \right\rceil, \bar{p}_{n-m}(J) + \bar{p}_{n-m+1}(J)\right\}$$

The upper bound U is the makespan of the schedule constructed according to the LPT rule. The computation of L and U requires $O(n)$ and $O(n\log n)$ time, respectively. Let $BPP(J)$ denote the resulting lower bound. Then, $BPP(J)$ can be computed in $O(n^2 \log U)$.

Consequently, a valid lower bound for the $P|r_j,q_j|C_{\max}$ is:

$$\min_{j\in J} r_j + BPP(J) + \min_{j\in J} q_j$$

This bound can be further improved by adding the sets $D_1'(J)$ and $D_2'(J)$ of dummy jobs described in Lemma 4.2 (cf. page 58). Let $J_{ext} = J \cup D_1'(J) \cup D_2'(J)$ denote the extended set of jobs. Clearly, we have:

$$BPP(J) \leq BPP(J_{ext})$$

Therefore, an improved $O(n^2 \log U)$ lower bound for the $P|r_j,q_j|C_{\max}$ is:

$$LB_4(J) = \min_{j\in J_{ext}} r_j + BPP(J_{ext}) + \min_{j\in J_{ext}} q_j$$

Obviously, any progress that would be made in the design of an improved lower bound for the bin packing problem could be advantageously used to improve LB_4.

Proposition 4.3

$LB_4(J) \geq LB_2(J)$

Proof. Clearly, $LB_4(J) \geq LB_1(J_{ext})$. Note that

$$LB_1(J_{ext}) = \min_{j \in J_{ext}} r_j + \left\lceil \frac{1}{m} \sum_{j \in J_{ext}} p_j \right\rceil + \min_{j \in J_{ext}} q_j$$

$$= \min_{j \in J} r_j + \left\lceil \frac{1}{m} \left(\sum_{j \in D'(J)} p_j + \sum_{j \in J} p_j + \sum_{j \in D(J)} p_j \right) \right\rceil + \min_{j \in J} q_j$$

$$= \left\lceil \frac{1}{m} \left(\sum_{h=1}^{m} \overline{r}_h(J) + \sum_{j \in J} p_j + \sum_{h=1}^{m} \overline{q}_h(J) \right) \right\rceil$$

$= LB_2(J)$ ■

Example 4.6: Consider the 10 job -2 machine instance defined by Table 4.7.

j	1	2	3	4	5	6	7	8	9	10
r_j	2	8	3	6	5	3	9	6	3	7
p_j	92	92	97	93	92	4	4	5	3	4
q_j	2	2	10	4	7	7	8	2	10	10

Table 4.7 - Data of the 10 job - 2 machine instance of example 4.6

We have $D'_1(J) = \{j_1 : r_{j_1} = 2,\ p_{j_1} = 1,\ q_{j_1} = 2\}$ and $D'_2(J) = \emptyset$. The lower and upper bounds computed for the corresponding $P||C_{\max}$ defined on $J_{ext} = J \cup \{j_1\}$ are $L = \max\left\{97, \left\lceil \frac{487}{2} \right\rceil, 93 + 92\right\} = 244$ and $U = 277$, respectively.

Consider the trial value $C = 275$. For $p = 92$, we have $J_1 = J_2 = \emptyset$ and $J_3 = \{1, 2, 3, 4, 5\}$. Thus, $BPP_2(C, p) = \max\left\{0, \left\lceil \frac{5}{\lfloor \frac{275}{92} \rfloor} \right\rceil\right\} = 3 > m$. Consequently, $C+1 = 276$ is a lower bound for the considered $P||C_{\max}$. Therefore, a valid lower bound for the $P|r_j, q_j|C_{\max}$ instance is $LB_4(J) = 2+276+2 = 280$ (Note that $LB_1(J) = 247$ and $LB_2(J) = 248$).

4.5 Job subset based lower bounds

In this section as well as in the two following ones, we introduce some *lifting procedures* that aim at enhancing a given lower bound. Although, we used a

similar terminology, the attention of the reader should be drawn to the fact that these procedures are not related to cutting-plane theory. Perhaps the simplest example of a lifting procedure is based on the observation that a valid lower bound on the optimal makespan of an instance defined on a jobset J is still valid when defined on a restricted subset S of jobs. Therefore, we can lift a given lower bound by taking its maximal value over all of the subsets.

For instance, by applying the job based lifting procedure to LB_1, we obtain the following valid bound:

$$JLB_1(J) = \max_{S \subseteq J} LB_1(S)$$

Carlier (1987) proved that JLB_1 can be computed in $O(n \log n)$ time. Indeed, JLB_1 is equal to the optimal makespan of the preemptive one machine problem $1|r_j, q_j, pmtn|C_{\max}$, obtained by replacing the m machines by a single one and dividing all the processing times by m. This problem is solved using Jackson's preemptive algorithm (see Carlier, 1982).

Also, it is worth noting that applying this lifting procedure to the bound LB_2 would yield, at best, the value of $JPPB$. Indeed, Carlier and Pinson (1998) proved that:

$$JPPB(J) = \max \left\{ LB_0(J), \max_{S \subseteq J} LB_2(S) \right\}$$

Clearly, since LB_0 is a trivial bound, then the computation of $JPPB$ is of interest only in the case of $\max_{S \subseteq J} LB_2(S) > LB_0(J)$. Carlier and Pinson (1998), have shown that in this case, the set J^* such that $LB_2(J^*) = \max_{S \subseteq J} LB_2(S)$ can be deduced from $JPPS$ as described below.

Let denote by θ the smallest decision time in $JPPS$ such that the schedule block ending at time θ is composed of more than m jobs and we have for some job j :

$$C_{\max}(JPPS) = \theta + a_j(\theta) + q_j$$

Let $J_\theta = \left\{ j_p \in J; C_{\max}(JPPS) = \theta + a_{j_p}(\theta) + q_{j_p} \right\}$. Jobs in J_θ are assumed to be indexed according to nondecreasing order of their tails. By performing backward from θ, consider the first decision time ω at which either a machine is idle, or the ending schedule block contains a job with complete tail less than $q_{j_{m+1}}$. The set J^* is the set of jobs having some part processed between ω and θ in $JPPS$.

Example 4.1(continued): Since $C_{\max}(JPPS) = 36 > LB_0(J) = 35$, then one can deduce the set J^* from $JPPS$. Note that $\theta = 24$ and $\omega = 13$. Thus $J^* = \{2, 3, 6, 7\}$.

The following proposition shows that $JPPB$ can be computed more efficiently if the subset $\overline{J}$ is considered instead of J.

Proposition 4.4

$$\max_{S \subseteq \overline{J}} LB_2(S) = \max_{S \subseteq J} LB_2(S).$$

Proof. Note that $LB_2(\overline{S}) \geq LB_2(S)$ for each set S containing any job of $J \backslash \overline{J}$. Therefore, there is necessarily a set of jobs $J^* \subseteq \overline{J}$ such that $LB_2(J^*) = \max_{S \subseteq J} LB_2(S)$. Since $J^* \subseteq \overline{J} \subseteq J$, then

$$LB_2(J^*) \leq \max_{S \subseteq \overline{J}} LB_2(S) \leq \max_{S \subseteq J} LB_2(S)$$

Consequently, by definition of J^*, we have $\max_{S \subseteq \overline{J}} LB_2(S) = \max_{S \subseteq J} LB_2(S)$. ■

The aforementioned subset J^* such that $LB_2(J^*) = \max_{S \subseteq \overline{J}} LB_2(S)$ can be used to approximate the bound $\max_{S \subseteq J} ALB_2(S)$ by computing the value $JALB_2(J) = ALB_2(J^*)$.

Since $LB_3(J) \geq LB_2(J)$ (cf. Proposition 4.2), then an immediate consequence is the following corollary :

Corollary 4.2

$$\max\{LB_0(J), LB_3(J^*)\} \geq JPPB(\overline{J})$$

Another consequence of Proposition 4.2 is that it would be interesting to compute the value of $\max_{S \subseteq \overline{J}} LB_3(S)$. We propose a greedy algorithm, described below, to compute an approximation of this value. The resulting bound will be denoted by JLB_3. Denote by S^* the set such that $LB_3(S^*) = JLB_3(J)$. In the computation of $JLB_3(J)$, the set S^* is initialized to $\overline{J}$. At each iteration, a particular job j_0 is removed from the current set S^*. The job j_0 is the job whose removal yields the most improvement to $LB_3(S^*)$. We iterate until no improvement is found. Note that it is useless to continue the procedure if $|S^*| \leq m + 1$. Indeed, each removal of a job will lead, at best, to a value equal to $LB_0(J)$. A formal description of the algorithm is as follows: The jobs $j_1, j_2, ..., j_{|S|}$ denote the different jobs of the set S.

Computation of $JLB_3(J)$

Step 0. *Set* $S^* = \overline{J}$ *and* $JLB_3(J) = LB_3(\overline{J})$.

Step 1.

1.1 *If* $|S^*| \leq m+1$, *then stop. Else, set* $k = 1$ *and* $LB = 0$.

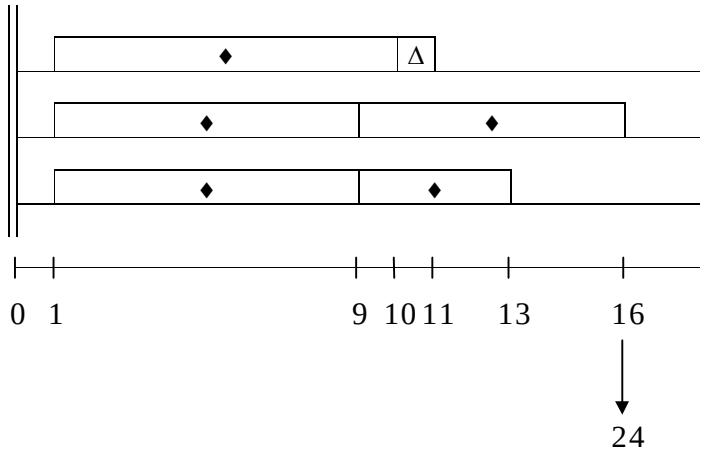
1.2 *Set* $S = S^* \backslash \{j_k\}$. *If* $LB_3(\overline{S}) > LB$, *then set* $LB = LB_3(\overline{S})$ *and* $j_0 = j_k$.

1.3 *Set* $k = k+1$. *If* $k \leq |S^*|$, *then go to step 1.2.*

Step 2. *If* $LB \leq JLB_3(J)$ *then stop. Else, set* $JLB_3(J) = LB$, $S^* = S^* \backslash \{j_0\}$ *and go to step 1.*

The bound $JLB_3(J)$ can be computed in $O(n^2 \log n)$ time.

Example 4.5 (continued): Let LB_3 be computed over the set $S = J\backslash\{7\}$. We have $\bar{S} = S$ and $D_2'(\bar{S}) = \emptyset$. In the corresponding $P, NC_{inc}||C_{\max}$ problem, the machine availabilities are $\{1, 1, 1\}$. For $\rho_1 = 1$, we have $n' = 5$, $|J_2| = 1$, $J_3 = \emptyset$ and $LB_3(S) = 24$ (see Figure 4.12a). This value corresponds to $JLB_3(J)$. Note that $LB_2(J) = JPPB(J) = 22$ and the optimal makespan is equal to 24 (see Figure 4.12b).



♦ *Jobs of* J_1 $\quad$ Δ *Jobs of* J_2

Figure 4.12a $JLB_3(J) = 24$

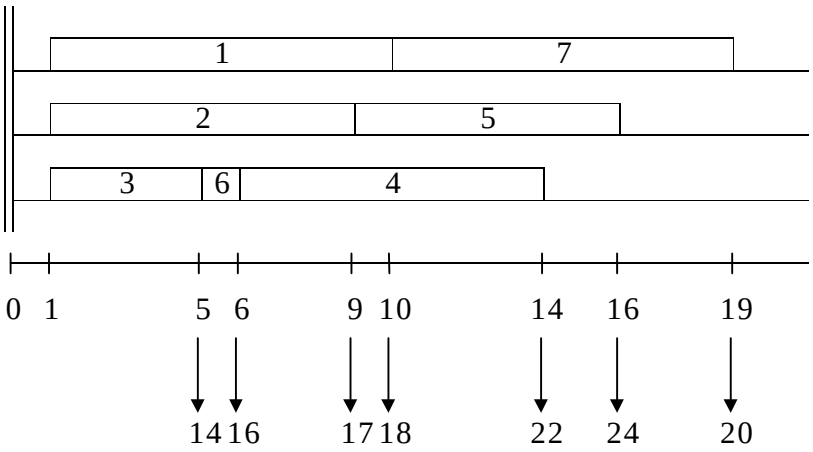


Figure 4.12b $C^*_{\max} = 24$

Let R and Q denote the number of distinct values of heads and tails, respectively. An $O(RQn^2 \log U)$ approximation of $\max_{S \subseteq J} LB_4(S)$, and denoted by $JLB_4(J)$, can be computed using the following algorithm:

Computation of $JLB_4(J)$

Set $JLB_4(J) = 0$.
For all pairs (r, q) *do*
    Compute the jobset $S = \{j \in J : r_j \geq r \text{ and } q_j \geq q\}$
    Set $JLB_4(J) = max\{JLB_4(J),\ r + BPP(S_{ext}) + q\}$
End (for)

4.6 Machine subset based lower bounds

In this section, we introduce a second basic lifting procedure which consists in considering only a restricted subset of machines. First, we state the following lemma provided by Blocher and Chand (1991).

Lemma 4.3

In any feasible schedule of a parallel machine problem with n jobs and m machines such that $n = \left\lfloor \frac{n}{m} \right\rfloor m + \rho$, *there is at least a set of* ρ *machines* $(1 \leq \rho \leq m-1)$ *which must process at least* $\rho \left(\left\lfloor \frac{n}{m} \right\rfloor + 1 \right)$ *jobs.*

In fact, we can go a step further and propose the following more general result.

Lemma 4.4

In any feasible schedule of a parallel machine problem with n jobs and m machines, there is at least a set of k machines ($1 \leq k \leq m$) which must process at least $\lambda_k(n) = k \left\lfloor \frac{n}{m} \right\rfloor + \min(k, \rho)$ *jobs, where* $\rho(n) = n - \left\lfloor \frac{n}{m} \right\rfloor m$.

Proof. Consider any feasible schedule and let α_i $(i = 1, ..., m)$ be the number of jobs processed by machine M_i. Assume with no loss of generality that $\alpha_1 \leq \alpha_2 \leq ... \leq \alpha_m$. Two cases have to be considered:

i) If $\alpha_{m-k} \leq \left\lfloor \frac{n}{m} \right\rfloor$: That is, the $m-k$ machines with minimum loads process at most $(m-k) \left\lfloor \frac{n}{m} \right\rfloor$ jobs. Thus, the remainder k machines process at least $n - (m-k) \left\lfloor \frac{n}{m} \right\rfloor = k \left\lfloor \frac{n}{m} \right\rfloor + \rho$ jobs.
ii) If $\alpha_{m-k} \geq \left\lfloor \frac{n}{m} \right\rfloor + 1$: Then, the k machines with highest loads process at least $k(\left\lfloor \frac{n}{m} \right\rfloor + 1) = k \left\lfloor \frac{n}{m} \right\rfloor + k$ jobs. ■

An immediate consequence of Lemma 4.4 is that a relaxed instance can be obtained by considering the instance with k machines and $\lambda_k(n)$ jobs, with data corresponding to the $\lambda_k(n)$ smallest processing times and k smallest release dates and delivery times $(k = 1, ..., m)$. This latter $P|r_j, q_j|C_{\max}$ instance will be denoted hereafter by P_k. It is worth noting that the data of P_k are uncoupled. Clearly, a whole family of lifted lower bounds is derived by considering each P_k $(k = 1, ..., m)$. Let $MLB_1^k(J)$, $MLB_2^k(J)$, $MLB_3^k(J)$ and $MLB_4^k(J)$ denote the respective values of LB_1, LB_2, LB_3, and LB_4 computed for the instance P_k. This yields the following valid lower bounds:

$$MLB_1(J) = \max_{1 \leq k \leq m} MLB_1^k(J)$$

$$MLB_2(J) = \max_{1 \leq k \leq m} MLB_2^k(J)$$

$$MLB_3(J) = \max_{1 \leq k \leq m} MLB_3^k(J)$$

$$MLB_4(J) = \max_{1 \leq k \leq m} MLB_4^k(J)$$

Interestingly, the bounds MLB_1 and MLB_2 can be improved in the following way. Note that, for fixed k and m, different values of n may yield

the same value of $\lambda_k(n)$. Let n_{λ_k} denote the minimal number of jobs yielding $\lambda_k(n)$. Consider the subset J_{λ_k} of the n_{λ_k} jobs having the largest processing times. Clearly, the following result holds:

Observation 4.4

1) $MLB_1^k(J_{\lambda_k}) \geq MLB_1^k(J)$
2) $MLB_2^k(J_{\lambda_k}) \geq MLB_2^k(J)$

The number of jobs n_{λ_k} can be computed using the following proposition.

Proposition 4.5

Given the integers m and k, the smallest number of jobs yielding $\lambda_k(n)$ is

$$n_{\lambda_k} = \lambda_k(n) + (m-k)(\left\lceil \frac{\lambda_k(n)}{k} \right\rceil - 1)$$

Proof. Consider the continuous function defined by

$$\Lambda_k(x) = k \left\lfloor \frac{x}{m} \right\rfloor + \min\left\{k, x - \left\lfloor \frac{x}{m} \right\rfloor m\right\}$$

We have:

$$\Lambda_k(x) = \begin{cases} x - \delta(m-k) & \text{if } \delta m \leq x \leq \delta m + k \\ k(\delta+1) & \text{if } \delta m + k \leq x < \delta m + m \end{cases} \qquad (\delta \in Z)$$

For a given λ, let x_λ denote the smallest x such that $\Lambda_k(x) = \lambda$. As it can be deduced from the plot of $\Lambda_k(.)$ (see Figure 4.13), we have $x_\lambda = \lambda + (m - k)(\left\lceil \frac{\lambda}{k} \right\rceil - 1)$. ■

Hence, we can systematically use the above observation to further lift the bounds MLB_1 and MLB_2 by taking

$$MLB_1(J) = \max_{1 \leq k \leq m} MLB_1^k(J_{\lambda_k})$$

$$MLB_2(J) = \max_{1 \leq k \leq m} MLB_2^k(J_{\lambda_k})$$

Although the computation of MLB_3^k and MLB_4^k on the set J_{λ_k} do not necessarily improve the bounds, our computational experiments showed that it is worth to use systematically J_{λ_k} in their computation.

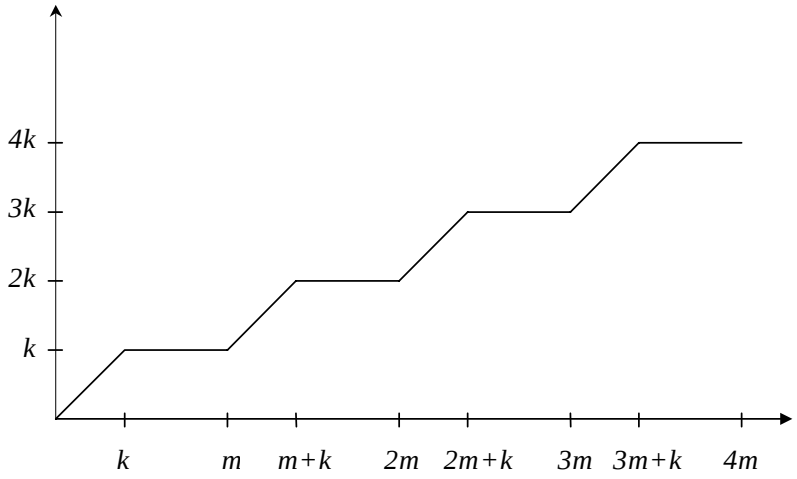


Figure 4.13. The plot of $\Lambda_k(.)$

Consequently, we take

$$MLB_3(J) = \max_{1 \le k \le m} MLB_3^k(J_{\lambda_k})$$

$$MLB_4(J) = \max_{1 \le k \le m} MLB_4^k(J_{\lambda_k})$$

The following example illustrates how LB_1 and LB_2 can be lifted using the machine subset lifting procedure.

Example 4.7: Consider the 11 job - 4 machine instance defined by Table 4.8.

j	1	2	3	4	5	6	7	8	9	10	11
r_j	9	6	2	9	8	2	2	5	7	8	5
p_j	95	96	80	99	97	91	96	92	96	93	95
q_j	9	4	1	8	8	6	7	5	8	3	5

Table 4.8 - Data of the 11 job - 4 machine instance of example 4.7

We have

$$LB_1(J) = 2 + \left\lceil \frac{1030}{4} \right\rceil + 1 = 261$$

$$LB_2(J) = \left\lceil \frac{11 + 1030 + 13}{4} \right\rceil = 264$$

For $k = 1$, we have $\lambda_1(n) = 3$, $n_{\lambda_1} = 9$ and

$$MLB_1^1(J_{\lambda_1}) = MLB_2^1(J_{\lambda_1}) = 2 + (92 + 93 + 95) + 3 = 285$$

For $k = 2$, we have $\lambda_2(n) = 6$, $n_{\lambda_2} = 10$ and

$$MLB_1^2(J_{\lambda_2}) = 2 + \left\lceil \frac{91 + 92 + 93 + 95 + 95 + 96}{2} \right\rceil + 3 = 286$$

$$MLB_2^2(J_{\lambda_2}) = \left\lceil \frac{4 + (91 + 92 + 93 + 95 + 95 + 96) + 7}{2} \right\rceil = 287$$

For $k = 3$, we have $\lambda_3(n) = 9$, $n_{\lambda_3} = 11$ and

$$MLB_1^3(J_{\lambda_3}) = 2 + \left\lceil \frac{80 + 91 + 92 + 93 + 95 + 95 + 96 + 96 + 96}{3} \right\rceil + 1 = 281$$

$$MLB_2^3(J_{\lambda_3}) = \left\lceil \frac{6 + (80 + 91 + 92 + 93 + 95 + 95 + 96 + 96 + 96) + 8}{3} \right\rceil = 283$$

For $k = 4$, we have clearly $MLB_1^4(J_{\lambda_4}) = LB_1(J) = 261$ and $MLB_2^4(J_{\lambda_4}) = LB_2(J) = 264$.

Consequently, the resulting machine subset based bounds are:

$$MLB_1(J) = \max\{285, 286, 281, 261\} = 286$$

$$MLB_2(J) = \max\{285, 287, 283, 264\} = 287$$

Let $F(l) = \sum_{j=1}^{l} \bar{p}_j(J)$ for $l = 1, ..., n$ and $F(0) = 0$. Since $F(l+1) = F(l) + \bar{p}_{l+1}(J)$, then the computation of $F(l)$ for all $l = 1, ..., n$ requires $O(n)$ time. Note that

$$MLB_1^k(J_{\lambda_k}) = \bar{r}_1(J_{\lambda_k}) + F(n - n_{\lambda_k} + \lambda_k(n)) - F(n - n_{\lambda_k}) + \bar{q}_1(J_{\lambda_k})$$

Thus, the computation of MLB_1 requires $O(n)$ time. Similarly, MLB_2 can be computed in $O(n)$ time. The bounds MLB_3 and MLB_4 require $O(mn \log n)$ and $O(mn^2 \log U)$ time, respectively.

4.7 Job-machine based lower bounds

The lower bounds derived with the machine based lifting procedure can be further improved if they are computed for every subset of J. In this section,

we propose for each lower bound computed in the previous section, a specific method to compute (exactly or approximately) its maximal value over all the subsets of jobs.

Firstly, we describe how to compute the lifted lower bound $JMLB_1(J) = \max_{S \subseteq J} MLB_1(S)$. For a given subset S, let $f_k(S) = \sum_{j=1}^{\lambda_k(|S|)} \bar{p}_j(S)$ $(k = 1, ..., m)$. We state the following result:

Proposition 4.6

The computation of $\max_{1 \leq k \leq m} \left\{ \max_{S \subseteq J} f_k(S) \right\}$ *requires* $O(n)$ *time.*

Proof. Let $S_h \subseteq J$ be the subset of h jobs with largest processing times among those of J $(h = 1, ..., n)$. Clearly, $\max_{S \subseteq J} f_k(S) = \max_{1 \leq h \leq n} f_k(S_h)$. Note that since $\lambda_m(h) = h$, then $f_m(S_h) \leq f_m(J)$ for all $h = 1, ..., n$. For $k = 1, ..., m-1$, two cases have to be considered:

i) If $h \in [\delta m, \delta m + k - 1]$ $(\delta \in Z)$: we have $\lambda_k(h+1) = \lambda_k(h) + 1$. Thus, $f_k(S_{h+1}) \geq f_k(S_h)$

ii) If $h \in [\delta m + k, \delta m + m - 1]$ $(\delta \in Z)$: we have $\lambda_k(h+1) = \lambda_k(h)$. Thus, $f_k(S_h) \geq f_k(S_{h+1})$

Consequently, $f_k(S_{\delta m+k}) \geq f_k(S_h)$ for all $h \in [\delta m, \delta m + m - 1]$ $(\delta \in Z)$. That is, the only values of h that are worth to be considered in the computation of $\max_{1 \leq h \leq n} f_k(S_h)$ are those satisfying $h \equiv k \pmod{m}$. Thus, for distinct values of k, we have distinct values of h. Moreover, there are $\left\lfloor \frac{n}{m} \right\rfloor$ values of h satisfying $h \equiv 0 \pmod{m}$ that need not to be considered**.** Consequently, the computation of $\max_{1 \leq k \leq m} \left\{ \max_{1 \leq h \leq n} f_k(S_h) \right\}$ requires only $n - \left\lfloor \frac{n}{m} \right\rfloor$ iterations.

Note that

$$f_k(S_h) = \sum_{j\,=\,n-h+1}^{n-h+\lambda_k(h)} \bar{p}_j(J) = F(n - h + \lambda_k(h)) - F(n - h)$$

Thus, $f_k(S_h)$ can be computed in $O(1)$ time. Therefore, $\max_{1 \leq k \leq m} \left\{ \max_{1 \leq h \leq n} f_k(S_h) \right\}$ can be computed in $O(n)$ time. ■

A consequence of Proposition 4.6 is that $JMLB_1(J)$ can be computed in $O(RQn)$ using the following algorithm:

Computation of $JMLB_1(J)$

Set $JMLB_1(J) = 0$.
For all pairs (r, q) *do*
Compute the jobset $S_{r,q} = \{j \in J : r_j \geq r \text{ and } q_j \geq q\}$
For all $h = m + 1, ..., |S_{r,q}|$ *do*
Set $JMLB_1(J) = max\{JMLB_1(J),\ LB_1(S_{r,q})\}$ *and* $k = h - \left\lfloor \frac{h}{m} \right\rfloor m$
If $k \neq 0$, *then set* $JMLB_1(J) = max\left\{JMLB_1(J),\ r + \left\lceil \frac{1}{k} f_k(S_h) \right\rceil + q\right\}$
End (for)
End (for)

The reader may have noticed that the values of $h = 1, ..., m$ have not been considered in the computation of $JMLB_1$ since they would yield at best LB_0.

A second interesting job-machine based lower bound is

$$JMLB_2(J) = \max_{S \subseteq J} MLB_2(S)$$

An $O(RQn)$ approximation of $JMLB_2$ can be computed by slightly modifying the algorithm used for $JMLB_1$. Indeed, it suffices to replace

$$r + \left\lceil \frac{1}{k} f_k(S_h) \right\rceil + q$$

by

$$\left\lceil \frac{1}{k} \left(\overline{r}_1(S_h) + \cdots + \overline{r}_k(S_h) + f_k(S_h) + \overline{q}_1(S_h) + \cdots + \overline{q}_k(S_h)\right) \right\rceil$$

Also, the bounds $\max_{S \subseteq J} MLB_3(S)$ and $\max_{S \subseteq J} MLB_4(S)$ can be computed approximately by computing $JMLB_3(J) = MLB_3^{k_0}(J^*)$ and $JMLB_4(J) = MLB_4^{k_0}(J^*)$, where J^* denotes the set such that $MLB_2(J^*) = JMLB_2(J)$ and $k_0 = |J^*| - \left\lfloor \frac{|J^*|}{m} \right\rfloor m$. The computations of $JMLB_3$ and $JMLB_4$ require $O(RQn + n \log n)$ and $O(RQn + n^2 \log U)$ time, respectively.

4.8 A computational study of the lower bounds

In order to assess the quality of the different lower bounds, we have generated 6 different random problem sets. The test problems have been designed with the aim of providing a good picture of the relative performance of the different lower bounds. The heads and tails of all jobs are drawn from the discrete uniform distribution on [1,10]. Each problem set is characterized by a number $\alpha \in \{0, 20, 40, 60, 80, 100\}$ which specifies that the processing times of $\alpha\%$ of the jobs of each instance of the set are drawn from the discrete uniform distribution on [90,100], and the processing times of the remaining jobs are drawn from the discrete uniform distribution on [1,5]. This type of processing times leads to different combinations of long/short processing times, and heads and tails. Each problem set contains 250 test problems corresponding to 10 instances for different combinations of n and m with $n \in \{20, 40, 60, 80, 100\}$ and $m \in \{2, 3, 5, 7, 9\}$. Therefore, 1500 test problems were generated. All the computational experiments were carried out on a PentiumIV 2 GHz Personal Computer with 256 MB RAM.

In Table 4.9, we report for each basic (i.e. non-lifted) lower bound the percentage of times it yields the maximal value over *all* of the bounds discussed in this chapter (Max), and the CPU time (in sec.) required for processing all of the 1500 instances ($Time$). We observe that the highest percentage (67.07%) is obtained for $SPLB$. Obviously, the high percentage indicate that this bound outperforms the remaining ones. However, we observe that LB_2, ALB_2, LB_3 and LB_4 perform quite well but require much less CPU time. It is worth noting that the two lower bounds $JPPB$ and PLB are found to be equal in all the generated instances, which is consistent with the preliminary results of Carlier and Pinson (1998).

	LB_0	LB_1	LB_2	ALB_2	$JPPB$	PLB	$SPLB$	LB_3	LB_4
Max	7.33	45.87	51.27	51.80	59.27	59.27	67.07	55.60	52.40
$Time$	0.01	0.01	0.01	0.22	5.09	6.51	6.67	0.43	0.01

Table 4.9 - Performance of the basic bounds

In Tables 4.10-4.12, we report the performance results of the lifted lower bounds. We see from Table 4.10 that the job subset lifting procedure improves the quality of all these bounds. The improvement is negligible for LB_1, LB_3 and LB_4 but more significant for ALB_2. Not surprisingly, these improvements caused a high increase of the CPU time. Table 4.11 shows that the machine subset lifting procedure is more effective, since the quality improvements are more important and the required CPU time is much less. Interestingly, we observe that this lifting procedure made MLB_2, MLB_3 and MLB_4 to outperform the best existing lower bounds ($JPPB$ and $JALB_2$). Indeed, MLB_3 requires 1.18 seconds (for all the 1500 instances) and yields for 64.73% of the instances the best value, whereas $JPPB$ (resp. $JALB_2$) requires 5.09 (resp. 6.75) seconds and is the best one for 59.27% (resp. 59.87%) of the instances. Table 4.12 provides evidence that combining the two lifting procedures yields the most effective lower bounds. Hence, $JMLB_3$ reached the maximal value over all the bounds for 89.27% of the instances and required a cumulative computing time of only 1.11 seconds. Moreover, even the lifted version of the (simple) lower bound LB_1 yields the remarkable percentage of 69.80% and requires just 0.45 seconds.

	JLB_1	$JALB_2$	JLB_3	JLB_4
Max	46.53	59.87	56.73	54.07
$Time$	0.17	6.75	40.87	3.23

Table 4.10 - Performance of the job subset based bounds

	MLB_1	MLB_2	MLB_3	MLB_4
Max	53.07	60.53	64.73	61.47
$Time$	0.06	0.09	1.18	0.30

Table 4.11 - Performance of the machine subset based bounds

	$JMLB_1$	$JMLB_2$	$JMLB_3$	$JMLB_4$
Max	69.80	85.40	89.27	85.47
$Time$	0.45	0.81	1.11	0.92

Table 4.12 - Performance of the job-machine subset based bounds

A more accurate picture of the respective performance of the different lower bounds is provided in Tables 4.13-4.15, where the detailed results are reported with respect to α, n, and m. We observe that our global analysis is generally consistent with the results displayed in these tables. However, we can emphasize the following points:

- The bound LB_4 is very competitive with the best existing (non-lifted) bounds for large number of jobs ($n \geq 80$), small number of machines ($m = 2, 3$), and in the presence of more jobs with large processing times ($\alpha \geq 60$).

- The effectiveness of the job subset lifting procedure decreases when there are more jobs with large processing times ($\alpha \geq 60$) and when the number of jobs increases ($n \geq 60$). Though, this lifting procedure performs slightly better on instances with a large number of machines ($m \geq 7$).

- The machine subset lifting procedure is effective when all the jobs have large processing times ($\alpha = 100$). Moreover, we observe that large improvements are achieved for instances with large number of machines.

- For $\alpha \leq 20$, we observe that $SPLB$ and $JALB_2$ yield the best results. For all other values of α, the bounds $JMLB_2$, $JMLB_3$ and $JMLB_4$ consistently outperform the other bounds.

	$\alpha = 0$	$\alpha = 20$	$\alpha = 40$	$\alpha = 60$	$\alpha = 80$	$\alpha = 100$
LB_0	24.00	16.00	4.00	0.00	0.00	0.00
LB_1	60.40	40.00	40.80	47.60	41.60	44.80
LB_2	72.80	43.20	44.00	52.40	43.60	51.60
ALB_2	74.00	43.20	44.00	54.40	43.60	51.60
$JPPB$	98.40	59.60	48.00	54.00	44.00	51.60
PLB	98.40	59.60	48.00	54.00	44.00	51.60
$SPLB$	98.80	82.40	60.00	62.00	47.60	51.60
LB_3	84.40	50.00	45.20	55.20	46.00	52.80
LB_4	72.80	45.20	44.40	53.60	45.60	52.80
JLB_1	63.20	40.80	40.80	47.60	42.00	44.80
$JALB_2$	100.00	59.60	48.00	55.60	44.00	52.00
JLB_3	86.40	52.40	46.80	55.20	46.80	52.80
JLB_4	74.00	50.00	46.00	54.80	46.80	52.80
MLB_1	60.40	40.00	40.80	47.60	46.80	82.80
MLB_2	72.80	43.20	44.00	52.40	51.20	99.60
MLB_3	84.40	50.00	45.20	55.20	54.00	99.60
MLB_4	72.80	45.20	44.40	53.60	53.20	99.60
$JMLB_1$	91.60	54.40	65.60	71.60	82.80	82.80
$JMLB_2$	74.00	66.40	85.60	90.80	96.00	99.60
$JMLB_3$	90.80	71.20	86.40	91.20	96.40	99.60
$JMLB_4$	74.00	66.40	86.00	90.80	96.00	99.60

Table 4.13 - Performance of the lower bounds with respect to α

	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
LB_0	24.00	10.33	2.33	0.00	0.00
LB_1	13.33	30.00	50.67	60.00	75.33
LB_2	25.67	35.33	56.67	62.00	76.67
ALB_2	26.33	35.67	57.67	62.67	76.67
$JPPB$	51.67	46.00	59.67	62.33	76.67
PLB	51.67	46.00	59.67	62.33	76.67
$SPLB$	71.67	55.33	66.33	65.33	76.67
LB_3	40.33	36.33	59.00	64.33	78.00
LB_4	26.67	35.67	57.67	64.00	78.00
JLB_1	16.00	30.33	51.00	60.00	75.33
$JALB_2$	53.33	46.33	60.00	63.00	76.67
JLB_3	45.67	36.67	59.00	64.33	78.00
JLB_4	32.00	37.33	58.33	64.67	78.00
MLB_1	22.67	37.00	57.00	66.67	82.00
MLB_2	42.00	45.33	63.33	68.67	83.33
MLB_3	57.00	46.33	65.67	71.00	83.67
MLB_4	43.00	45.67	64.33	70.67	83.67
$JMLB_1$	33.00	57.00	75.67	87.33	96.00
$JMLB_2$	57.67	81.67	91.33	96.33	100.00
$JMLB_3$	72.67	84.67	92.33	96.67	100.00
$JMLB_4$	58.00	81.67	91.33	96.33	100.00

Table 4.14 - Performance of the lower bounds with respect to n

	$m = 2$	$m = 3$	$m = 5$	$m = 7$	$m = 9$
LB_0	0.00	0.67	7.00	10.00	19.00
LB_1	90.33	69.33	44.00	17.00	8.67
LB_2	98.67	73.67	51.33	19.33	13.33
ALB_2	99.00	74.00	51.67	20.00	14.33
$JPPB$	99.67	75.33	58.67	29.67	33.00
PLB	99.67	75.33	58.67	29.67	33.00
$SPLB$	99.67	79.00	65.00	42.67	49.00
LB_3	99.67	75.67	53.67	24.33	24.67
LB_4	98.67	74.33	52.33	20.33	16.33
JLB_1	92.00	69.67	44.67	17.33	9.00
$JALB_2$	100.00	76.33	59.33	30.33	33.33
JLB_3	99.67	75.67	54.67	27.00	26.67
JLB_4	99.67	76.33	54.00	22.67	17.67
MLB_1	90.33	75.33	47.33	28.67	23.67
MLB_2	98.67	80.33	54.67	39.00	30.00
MLB_3	99.67	82.33	57.00	44.33	40.33
MLB_4	98.67	81.00	55.67	40.00	32.00
$JMLB_1$	92.00	87.00	71.33	55.67	43.00
$JMLB_2$	99.67	96.67	87.33	77.67	65.67
$JMLB_3$	99.67	96.67	91.33	83.33	75.33
$JMLB_4$	99.67	96.67	87.67	77.67	65.67

Table 4.15 - Performance of the lower bounds with respect to m

Interestingly, we found that a very effective lower bound is

$$LB^* = \max\{SPLB, JMLB_3\}$$

Indeed, this bound is the best one for 99.47% of the instances. We recall that the semi-preemptive bound $SPLB$ performs a bisection search on an interval $[C^-, C^+]$. Therefore, for an efficient computation of LB^*, it suffices to take $C^- = JMLB_3$ (instead of $JPPB$) in the computation of $SPLB$. It is worth noting that the obtained schedule may not be an optimal semi-preemptive one. However, it yields a strong lower bound on the optimal nonpreemptive makespan. The obtained bound LB^* requires only 3.92 sec. for all the 1500 test problems. Moreover, as it can be appreciated from Table 4.16, this latter

bound seems to be little sensitive to the variation of α, n and m.

		Max
α	0	98.80
	20	100.00
	40	100.00
	60	98.40
	80	100.00
	100	99.60

		Max
n	20	98.67
	40	99.67
	60	99.67
	80	99.33
	100	100.00

		Max
m	2	99.67
	3	99.33
	5	99.33
	7	99.33
	9	99.67

Table 4.16 - Performance of LB^* with respect to α, n and m

We performed additional experiments with the objective of evaluating the impact of large variation of heads and tails. For that purpose, we generated a set of instances with the following properties. The number of jobs is taken equal to 20, 40, 60, 80 and 100. The number of machines is taken equal to 2, 3, 5, 7 and 9. The heads and tails are drawn from the discrete uniform distribution on [1,100]. The processing times are drawn from the discrete uniform distribution on [90,100]. For each combination of (n, m), 10 instances have been randomly generated. It is worth noting that this way of generating instances guaranties that no large job sizes completely dominate the contribution of the smaller ones. Table 4.17 depicts the results obtained on this set of instances. We see that these new results are consistent with our previous analysis.

	Max	$Time$
LB_0	0.40	0.01
LB_1	3.20	0.01
LB_2	43.60	0.01
ALB_2	52.00	0.03
$JPPB$	44.00	4.06
PLB	44.00	4.92
$SPLB$	44.40	4.98
LB_3	44.00	0.20
LB_4	44.00	0.01
JLB_1	3.20	0.01
$JALB_2$	52.40	7.82
JLB_3	44.00	11.46
JLB_4	44.40	18.31
MLB_1	12.80	0.01
MLB_2	87.60	0.01
MLB_3	87.60	0.39
MLB_4	87.60	0.06
$JMLB_1$	15.60	1.48
$JMLB_2$	90.80	2.57
$JMLB_3$	90.80	2.71
$JMLB_4$	90.80	2.51
LB^*	91.60	4.25

Table 4.17 - Impact of large variation of heads and tails

Chapter 5

A time-windows-based branch-and-bound algorithm for the $P|r_j, q_j|C_{\max}$

Polynomial algorithms for finding an optimal solution to an $\mathcal{NP}$-hard combinatorial optimization problem are not likely to exist. However, there exist several techniques for solving these problems to optimality (such as integer programming, branch-and-bound, dynamic programming, etc...). Such techniques are generally based on enumerating all the feasible solutions. The branch-and-bound method is a typical technique of intelligent enumeration. Although this method is exponential, it can be empirically very efficient. For instance, the branch-and-bound algorithm proposed by Carlier (1982) for the one machine scheduling problem $1|r_j, q_j|C_{\max}$ is able to solve very large instances in a small computing time. Moreover, it has been successfully embedded as a basic component in several algorithms for solving more complex scheduling problems (Adams et al., 1988).

The branch-and-bound technique is a recursive process which roughly consists in two phases. Firstly, the set of feasible solutions is partitioned into several subsets. This phase is referred to as the *branching* phase. Then, the subsets which are identified not to include any optimal solution are excluded. For a minimization problem, this can be performed if computing a lower bound on the considered subset yields a value which is larger than or equal to a given upper bound on the optimal solution. This phase is referred to as the *bounding* phase. The branch-and-bound method successively applies branching and bounding in exploring the global set of feasible solutions and finally leads to at least one optimal solution. The whole process can be more simply represented by a solution tree where each set of feasible solutions

is represented by a node, and its subsets are represented by its descendant nodes. The efficiency of a branch-and-bound algorithm relies essentially on its branching scheme and on the tightness of the used bounds. A more detailed description of branch-and-bound algorithms is provided in several textbooks including Nabeshima et al. (1982), Minoux (1986), and Nemhauser and Wolsey (1988).

In this chapter, we develop a new branch-and-bound algorithm for the $P|r_j, q_j|C_{\max}$. With each job is associated a time interval in which it has to be processed. To branch, a node of the tree with the smallest lower bound is selected, and the time interval of a particular job is divided into two smaller ones. This branching rule has been first introduced by Carlier (1987) whose algorithm is detailed in Section 5.3. Extensive experimental analysis shows that our algorithm consistently outperforms Carlier's one. It is worth noting that most of the material presented in this chapter has been published in Gharbi and Haouari (2002).

5.1 Data representation

With each node N of the branch-and-bound tree is associated a data set containing for each job j its processing time p_j, a lower bound $LB(N)$, an upper bound $UB(N)$, and a time window $[r_j(N), d_j(N)]$, where $r_j(N)$ and $d_j(N)$ represent the release date and the deadline of job j, respectively. We denote by UB the minimum value of $UB(N)$ over all nodes of the tree. The deadline is initially set to $d_j(N) = UB - q_j(N) - 1$, where $q_j(N)$ is the tail of job j. Clearly, any feasible schedule has a makespan less than UB if the jobs are completed before their respective deadlines. Node N represents the feasible schedules which satisfy the condition that each job is processed within its corresponding time window. Moreover, the application of the preprocessing algorithm to the instance defined by the node N, generates three sets $J_R(N)$, $J_Q(N)$ and $\overline{J}(N)$ (following the notation of chapter 3, page 29).

5.2 Upper bounds

For minimization problems, computing a tight upper bound on the optimal solution can have a significant impact on the performance of branch-and-bound algorithms. For the $P|r_j, q_j|C_{\max}$, the makespan of any approximate

schedule is an upper bound on the optimal makespan. The only existing approximate algorithm for the $P|r_j, q_j|C_{\max}$ is that proposed by Jackson (1955). In this section, we describe Jackson's algorithm and we introduce some new variants which yield a better worst-case ratio.

5.2.1 Jackson's algorithm

Jackson's algorithm provides a simple but efficient approximation of the $P|r_j, q_j|C_{\max}$ optimal solution. This algorithm is based on a dispatching rule which schedules the available job with the largest tail at the earliest available machine. In the following description of Jackson's algorithm, the set J denotes the set of jobs that have to be scheduled, I denotes the set of jobs that have not been scheduled yet, t_j denotes the starting time of job j, and $u_1, u_2, ..., u_m$ denote the machine availabilities.

Jackson's algorithm

Step 0. *Set* $u_1 = u_2 = ... = u_m = 0$; $I = J$.

Step 1. *If* I *is empty, then stop.*

Step 2. *Let* m_0 *be the earliest available machine. If* $r_j > u_{m_0}$ *for all* $j \in I$, *then set* $u_{m_0} = \min_{j \in I} r_j$.

Step 3. *Let* j_0 *be the job in* I *such that* $r_{j_0} \leq u_{m_0}$ *and* $q_{j_0} = \max_{j \in I} q_j$. *Set* $t_{j_0} = u_{m_0}$ *and assign job* j_0 *to machine* m_0. *Set* $I = I \setminus \{j_0\}$ *and* $u_{m_0} = u_{m_0} + p_{j_0}$. *Go to Step 1.*

Ranking jobs in nondecreasing (respectively nonincreasing) order of their r_j (respectively q_j) takes $O(n \log n)$ time. Therefore, the complexity of Jackson's algorithm is $O(n \log n)$.

Example 5.1 : Figure 5.1 shows the Jackson's schedule obtained for a 5 job - 2 machine instance which data are given by Table 5.1. Its makespan is equal to 24.

j	1	2	3	4	5
r_j	9	7	5	1	2
p_j	6	10	6	6	9
q_j	2	1	8	9	6

Table 5.1. Data of the 5 job - 2 machine instance of example 5.1

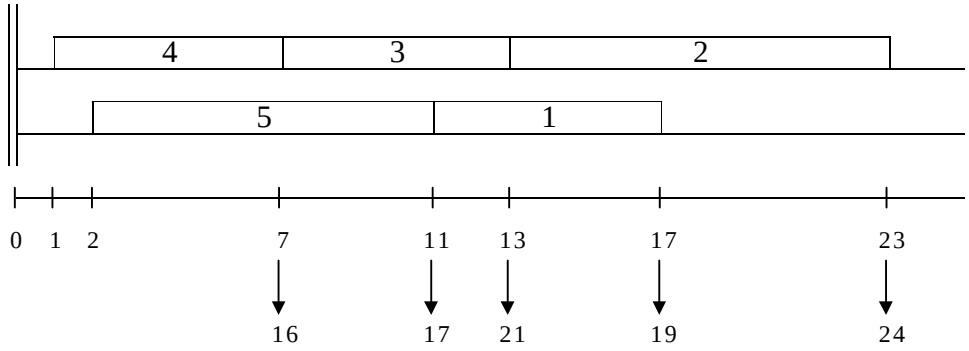


Figure 5.1. Example of Jackson′s schedule

5.2.2 New variants of Jackson's algorithm

Since the $P|r_j, q_j|C_{\max}$ problem is symmetric, a second Jackson's rule based approximation schedule can be derived by considering the Backward problem. This schedule, denoted by the Backward Jackson's schedule, may be better than Jackson's schedule.

Example 5.1 (continued): Figure 5.2 shows the Backward Jackson's schedule constructed for the instance defined by Table 5.1. Its makespan is equal to 22.

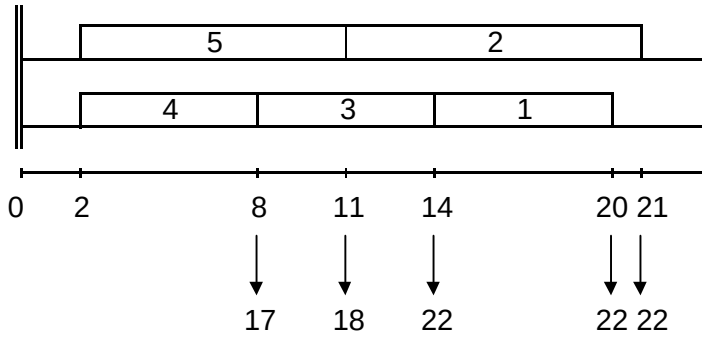


Figure 5.2. Example of Backward Jackson′s schedule

A second variant consists in applying Jackson's algorithm to the set $\bar{J}$ and to complete the schedule with $GSCA$ (cf. page 30). It is worth noting that, for example 3.1 (cf. page 29), this variant yields a makespan equal to 25, whereas the basic Jackson's algorithm applied to J yields a makespan equal to 27.

The following theorems and corollaries provide an upper bound on the maximal deviation of this variant of Jackson's algorithm. These results improve those of Gusfield (1984) and Carlier (1987) (cf. page 17):

Theorem 5.1

Assume that the preprocessing algorithm is applied to J and that Jackson's schedule is applied to $\overline{J}$. Then, the GSCA provides a schedule with makespan $C_{\max}(J)$ such that

$$C_{\max}(J) - C^*_{\max}(J) \leq \left\lceil \frac{2m-1}{m} \max_{j \in \overline{J}} p_j \right\rceil - 1$$

and this bound is tight.

Proof. First, we recall that in any schedule, a critical job denotes a job which completion time is equal to the makespan of the schedule. Two cases are considered :

i) There is no critical job $j_0 \in \bar{J}$ in the $GSCA$ schedule : that is, $C_{\max}(J) > C_{\max}(\bar{J})$ and therefore $C_{\max}(J) = \max_{J_R \cup J_Q} (r_j + p_j + q_j)$. Consequently, $GSCA$ provides the optimal schedule and $C_{\max}(J) - C^*_{\max}(J) = 0$.

ii) There is a critical job $j_0 \in \bar{J}$ in the $GSCA$ schedule : that is, $C_{\max}(J) = C_{\max}(\bar{J})$. Gusfield (1984) proved that $C_{\max}(\bar{J}) - C^*_{\max}(\bar{J}) < \dfrac{2m-1}{m} \max_{j \in \overline{J}} p_j$. That is,

$$C_{\max}(\bar{J}) - C^*_{\max}(\bar{J}) \leq \left\lceil \frac{2m-1}{m} \max_{j \in \overline{J}} p_j \right\rceil - 1$$

Since we have $C^*_{\max}(\bar{J}) \leq C^*_{\max}(J)$, then

$$C_{\max}(J) - C^*_{\max}(J) \leq \left\lceil \frac{2m-1}{m} \max_{j \in \overline{J}} p_j \right\rceil - 1$$

Now, consider the 13 job - 2 machine instance which data is given by Table 5.2.

j	1	2	3	4	5	6	7	8	9	10	11	12	13
r_j	8	8	9	9	9	9	9	8	8	8	0	5	9
p_j	4	4	2	2	2	2	4	1	1	1	6	5	7
q_j	9	9	19	19	19	19	19	7	7	7	9	1	0

Table 5.2. Data of a 13 job - 2 machine instance

After preprocessing, we have $J_R = \{11\}$, $\overline{J} = \{1, ..., 10\}$, and $J_Q = \{12, 13\}$. Figure 5.3a shows the schedule provided by $GSCA$ after applying Jackson's algorithm to $\overline{J}$. Its makespan is equal to 39.

Consider the set $S = \{3, 4, 5, 6, 7\}$, we have $LB_2(S) = 34$. Therefore the schedule shown in Figure 5.3b is optimal for this instance. The deviation of the makespan provided by $GSCA$ from the optimal one is equal to $5 = \left\lceil \frac{2m-1}{m} \max_{j \in \overline{J}} p_j \right\rceil - 1$. ■

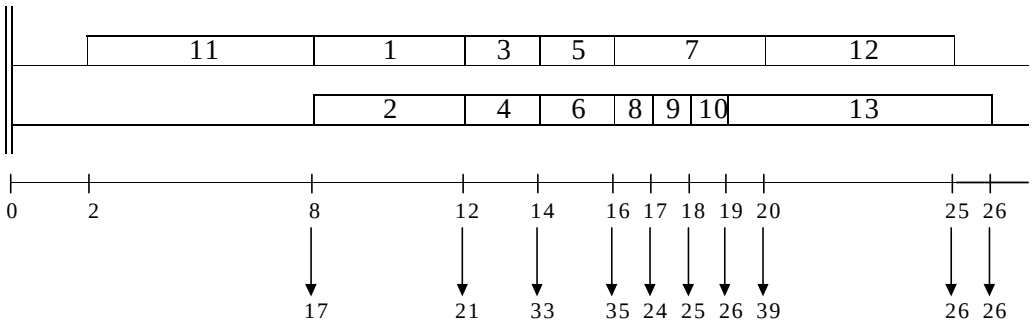


Figure 5.3a. $GSCA$ schedule

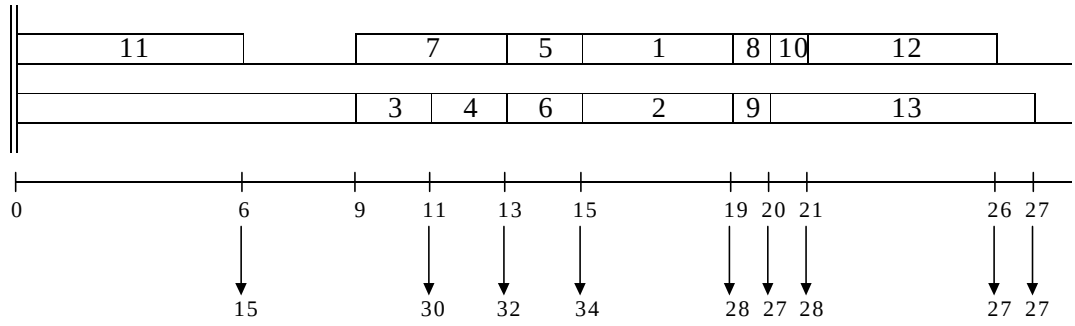


Figure 5.3b. An optimal schedule

Theorem 5.2

Assume that the preprocessing algorithm is applied to J and that Jackson's schedule is applied to $\overline{J}$. Then, the GSCA provides a schedule with makespan $C_{\max}(J)$ such that

$$C_{\max}(J) - C^*_{\max}(J) \leq 2(\max_{j \in \overline{J}} p_j - 1)$$

and this bound is tight.

Proof. The proof is based on the following result (Carlier, 1987) :

$$JS(J) - LB_1(J_0) \leq 2(\max_{j \in J} p_j - 1)$$

where $JS(J)$ denotes the makespan of Jackson's schedule obtained on J, and J_0 is the so called critical set defined as follows. Let j_0 be a critical job in Jackson's schedule. First, construct the subset J_1 of J, which contains all jobs with tail greater than or equal to q_{j_0}. Then, determine the set I^* for which $LB_1(I^*) = JLB_1(J_1)$ (see Carlier, 1982). The set J_0 is composed of all jobs of J_1 with head greater than or equal to $\min_{j \in I^*} r_j$.

Note that $LB_1(J_0) \leq JLB_1(J)$. It is easy to see that

$$JLB_1(J) \leq JPPB(J) = \max \left\{ LB_0(J), \max_{S \subseteq J} LB_2(S) \right\}$$

Indeed, denote by J^* the set such that $LB_1(J^*) = JLB_1(J)$. Clearly, if $|J^*| > m$, then $LB_1(J^*) \leq LB_2(J^*) \leq \max_{S \subseteq J} LB_2(S)$. Otherwise, we have

$$LB_1(J^*) \leq C^*_{\max}(J^*) = LB_0(J^*) \leq LB_0(J)$$

Consequently, for any set of jobs S, we have

$$JS(S) - JPPB(S) \leq 2(\max_{j \in S} p_j - 1)$$

We showed that $C_{\max}(J) = \max \left\{ JS(\overline{J}), LB_0(J_R), LB_0(J_Q) \right\}$ (cf. page 31). Two cases are considered :

i) If $C_{\max}(J) = \max \left\{ LB_0(J_R), LB_0(J_Q) \right\}$, then the $GSCA$ provides the optimal schedule. That is, $C_{\max}(J) - C^*_{\max}(J) = 0$.

ii) If $C_{\max}(J) = JS(\overline{J})$, then we have :

$$\begin{aligned} C_{\max}(J) - C^*_{\max}(J) &= JS(\overline{J}) - C^*_{\max}(J) \\ &\leq JS(\overline{J}) - JPPB(\bar{J}) \\ &\leq 2(\max_{j\in\overline{J}} p_j - 1). \end{aligned}$$

Now, consider the 13 job - 3 machine instance which data is given in Table 5.3.

j	1	2	3	4	5	6	7	8	9	10	11	12	13
r_j	0	0	0	0	1	1	1	1	1	1	1	0	0
p_j	3	3	3	1	1	1	1	1	1	1	3	10	14
q_j	16	16	16	16	20	20	20	20	20	20	20	3	1

Table 5.3. Data of a 13 job - 3 machine instance

After preprocessing, we have $J_R = \emptyset$, $\overline{J} = \{1, ..., 11\}$, and $J_Q = \{12, 13\}$. Figure 5.4a shows the schedule provided by $GSCA$ after applying Jackson's algorithm to $\overline{J}$. Its makespan is equal to 28. Since $LB_0(J) = 24$, then the schedule shown in figure 5.4b is optimal for this instance.

The deviation of the makespan provided by $GSCA$ from the optimal one is equal to $4 = 2(\max_{j\in\overline{J}} p_j - 1)$. ■

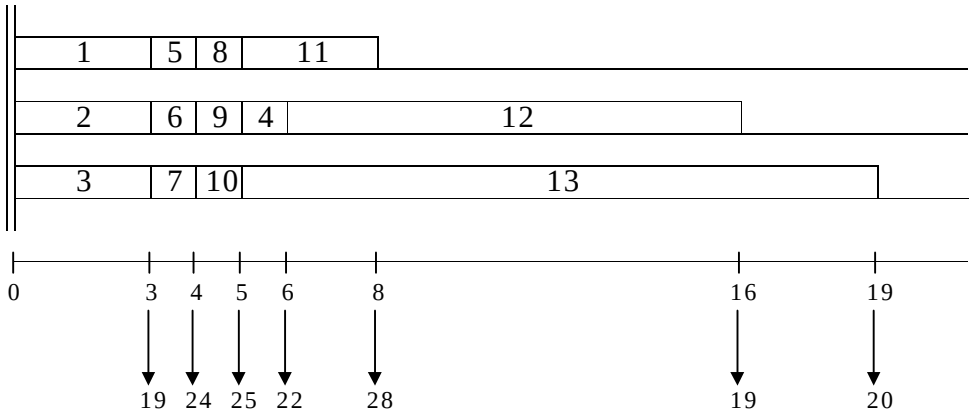


Figure 5.4a. $GSCA$ schedule

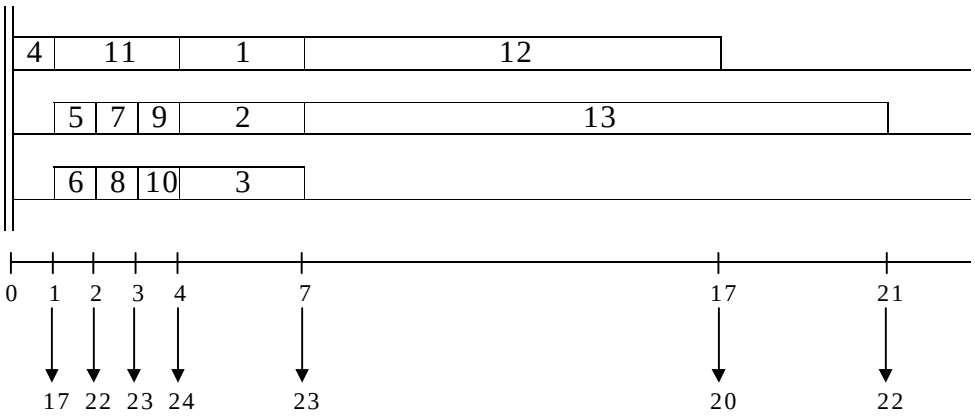


Figure 5.4b. An optimal schedule

Corollary 5.1

Assume that, after preprocessing, all processing times in $\overline{J}$ are equal to one. Then, after applying Jackson's algorithm to $\overline{J}$, the GSCA provides an optimal schedule.

Corollary 5.2.

The deviation of the lower bound $JPPB(\bar{J})$ from the optimal makespan is less than or equal to $2(\max_{j\in\overline{J}} p_j - 1)$.

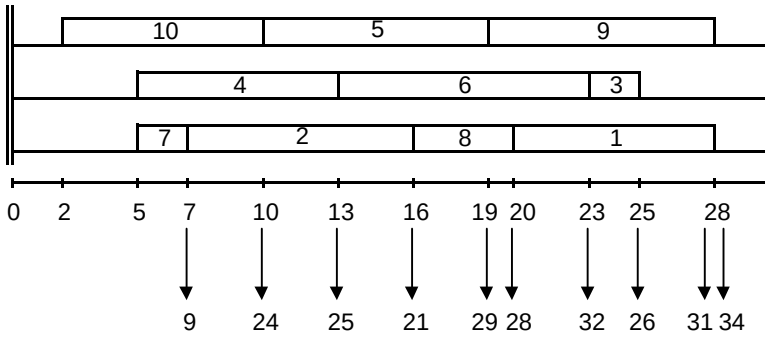
It is worth noting that the makespan of the schedule constructed by $GSCA$ after applying Jackson's algorithm to $\overline{J}$, is not necessarily less than or equal to that of Jackson's schedule obtained on J (as it can be appreciated by the following example). However, there is no dominance between the two approximations.

Clearly, a third variant of Jackson's algorithm consists in applying its Backward version to $\bar{J}$ and complete the schedule with $GCSA$. In the sequel, we denote by $JS(I)$ (respectively, $JS^{-1}(I)$), the makespan of the schedule obtained by $GSCA$ after Jackson's schedule (respectively, the Backward Jackson's schedule) has been constructed for a set of jobs $I \subseteq J$.

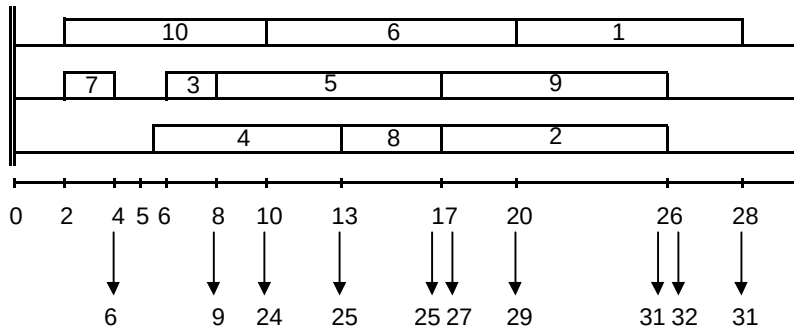
Example 5.2: Consider the 10 job - 3 machine instance defined by Table 5.4. We have $J_R = \{7\}$, $J_Q = \{3\}$ and $\bar{J} = J\backslash\{3, 7\}$. The schedule constructed by $GSCA$ after applying Jackson's algorithm to $\bar{J}$ yields a makespan equal to 34 (see figure 5.5a), whereas Jackson's schedule obtained on J yields a makespan equal to 32 (see figure 5.5b).

j	1	2	3	4	5	6	7	8	9	10
r_j	13	7	6	5	8	10	2	9	11	2
p_j	8	9	2	8	9	10	2	4	9	8
q_j	3	5	1	12	10	9	2	8	6	14

Table 5.4. Data of the 10 job - 3 machine instance of example 5.2



5.5a. $JS(\bar{J}) = 34$



5.5b. $JS(J) = 32$

Figure 5.5. $JS(J) < JS(\bar{J})$

5.3 Carlier's algorithm

The only exact algorithm proposed so far for the $P|r_j, q_j|C_{\max}$ is Carlier's one. In this section, we review Carlier's branch-and-bound algorithm. The basic ideas behind this algorithm will be used later as a starting point for our algorithm.

In his branch-and-bound algorithm, Carlier used the lower bound $LB_2(J_0)$, where J_0 is the subset of J defined in page 88. Even if Carlier did not explicitly mention the use of LB_0 in his branch-and-bound algorithm, we will

refer to the bound $\max\{LB_0(J), LB_2(J_0)\}$ as Carlier's lower bound. In the sequel, this bound will be denoted by $CLB(J)$. The upper bound $UB(N)$ is the value of the makespan of Jackson's schedule obtained for the instance defined by node N.

5.3.1 Branching rules

Let N be the node of the tree with minimal lower bound. The optimal makespan remains unchanged if the completion time of any job is set equal to $LB(N)$. Then, the tail $q_j(N)$ of every job j can be adjusted as follows :

$$q_j(N) = \max\{q_j(N), LB(N) - d_j(N)\} \tag{5.1}$$

If a better upper bound UB is found, all deadlines have to be adjusted as follows :

$$d_j(N) = \min\{d_j(N), UB - q_j(N)\} \tag{5.2}$$

For each job j is defined a margin M_j such that :

$$M_j = d_j(N) - r_j(N) - p_j$$

Note that in order to process a job j within its time window, its start time t_j must lie in the interval $I_j = [r_j(N), d_j(N) - p_j]$. For branching, a job j^* is selected (the selection criterion is discussed below), and the corresponding interval I_{j^*} is split into two disjoint intervals $I^1_{j^*} = [r_{j^*}(N), r_{j^*}(N) + \lceil M_{j*}/2 \rceil - 1]$ and $I^2_{j^*} = [r_{j^*}(N) + \lceil M_{j*}/2 \rceil, d_{j^*}(N) - p_{j^*}]$. Thus, two new nodes N_1 and N_2 are created. All the data of jobs in $J \setminus \{j^*\}$ are copied from node N to nodes N_1 and N_2.

For node N_1 we define :

$$r_{j*}(N_1) = r_{j*}(N)$$
$$d_{j*}(N_1) = r_{j*}(N) + \lceil M_{j*}/2 \rceil + p_{j*} - 1$$
$$q_{j*}(N_1) = \max\{q_{j*}(N), LB(N) - d_{j*}(N_1)\}$$
$$LB(N_1) = \max\{LB(N), CLB(N_1)\}$$

For node N_2 we define :

$$r_{j*}(N_2) = r_{j*}(N) + \lceil M_{j*}/2 \rceil$$

$$d_{j*}(N_2) = d_{j*}(N)$$

$$q_{j*}(N_2) = q_{j*}(N)$$
$$LB(N_2) = \max\{LB(N), CLB(N_2)\}$$

Figure 5.6 shows the relation between the intervals in which job j^* has to be processed for nodes N_1 and N_2.

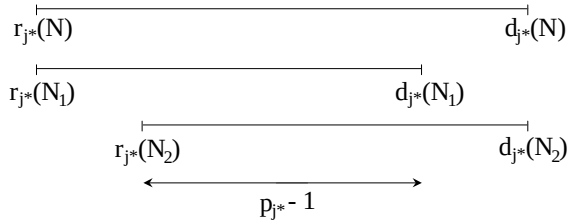


Figure 5.6. Time intervals of job j^* for nodes N, $N_1$and N_2.

Criteria for selecting j^*

Three selecting rules have been suggested :

$\mathbf{R}_1$: If there are more than m jobs in the critical set J_0, job j^* is selected in J_0 such that the values of $LB_2(J_0)$ for the nodes N_1 and N_2 are as large as possible. This lower bound will increase when j^* is such that $r_{j^*}(N) \leq \overline{r}_m(J_0)$ or $q_{j^*}(N) \leq \overline{q}_m(J_0)$.

The increase of $LB_2(J_0)$ for node N_1 is equal to

$$\lambda_{j^*} = \begin{cases} \min\left\{\overline{q}_{m+1}(J_0), q'_{j^*}\right\} - q_{j^*}(N) & \text{if } q_{j^*}(N) \leq \overline{q}_m(J_0), \\ 0 & \text{otherwise,} \end{cases}$$

where $q'_{j^*} = \max\{q_{j^*}(N), LB(N) - d_{j^*}(N_1)\}$.

The increase of $LB_2(J_0)$ for node N_2 is equal to

$$\mu_{j^*} = \begin{cases} \min\{\overline{r}_{m+1}(J_0), r_{j^*}(N_2)\} - r_{j^*}(N) & \text{if } r_{j^*}(N) \leq \overline{r}_m(J_0), \\ 0 & \text{otherwise.} \end{cases}$$

Job j^* is selected such that $\lambda_{j^*} + \mu_{j^*}$ is maximal. However, two cases may occur. Either $\lambda_j + \mu_j = 0$ for all $j \in J_0$, or J_0 does not contain more than m elements. In these cases, job j^* is selected according to R_2.

R_2- Let j_0 be a critical job of Jackson's schedule. Assume that it is executed by machine m_0. Perform backward from j_0 and let job j_1 be the first job processed by m_0 and starting at its release date**.** The critical path is defined as the set containing j_0, j_1 and all jobs in between that are processed by m_0. Consider the critical path of Jackson's schedule for the data set of node N. Job j^* is selected as the job on the critical path with smallest tail if this tail is smaller than q_{j_0}. If no such job exists, then j^* is selected according to R_3.

R_3- Select the job with the largest margin.

Remark: It is noteworthy that if job j^* is selected according to R_1 or R_2 (i.e. not as the job with the largest margin), then one must make sure that M_{j^*} is not zero. Otherwise, nodes N_1, N_2 and N will be identical and therefore, the algorithm will stall.

Finiteness of the branch-and-bound algorithm : When all margins are zero, the schedule is fixed and it is easy to check its feasibility. When a margin is negative, then there is no feasible schedule. All margins become zero or negative after at most $\sum_{j \in J} \lceil \log_2(M_j + 1) \rceil$ branchings, where M_j $(j \in J)$ are the initial margins. Consequently, the method is finite.

5.3.2 Description of the algorithm

Step 0. *Make a root node N containing the data set of a $P|r_j, q_j|C_{\max}$ problem. Set $UB = UB(N)$, $LB(N) = CLB(N)$, and $d_j(N) = UB - q_j(N)$ for all j.*

Step 1. *Select the node N of the tree with minimal lower bound. If*

$LB(N) \geqslant UB$, then stop.

Step 2. *Adjust $q_j(N)$ and $d_j(N)$ using equations (5.1) and (5.2) for all j. If a margin is negative, then erase N and go to step 1.*

Step 3. *If all margins are not zero, then set $LB(N) = \max\{LB(N),$*

$CLB(N)\}$ and go to step 4. Else, if the schedule is feasible, then set $UB = \min\{UB, UB(N)\}$, erase N and go to step 1.

Step 4. *Select j^* and create nodes N_1 and N_2. Compute $UB(N_1)$ and $UB(N_2)$. Set $UB = \min\{UB, UB(N_1), UB(N_2)\}$. Erase N and go to step 1.*

Example 5.3: Figure 5.7 shows the tree developed by Carlier's algorithm for the instance given by Table 5.5.

j	1	2	3	4	5	6	7	8	9	10
r_j	4	5	1	17	23	17	17	16	11	1
p_j	4	7	9	10	10	1	9	8	7	8
q_j	8	3	11	22	17	9	9	2	24	25

Table 5.5. Data of the 10 job - 2 machine instance of example 5.3

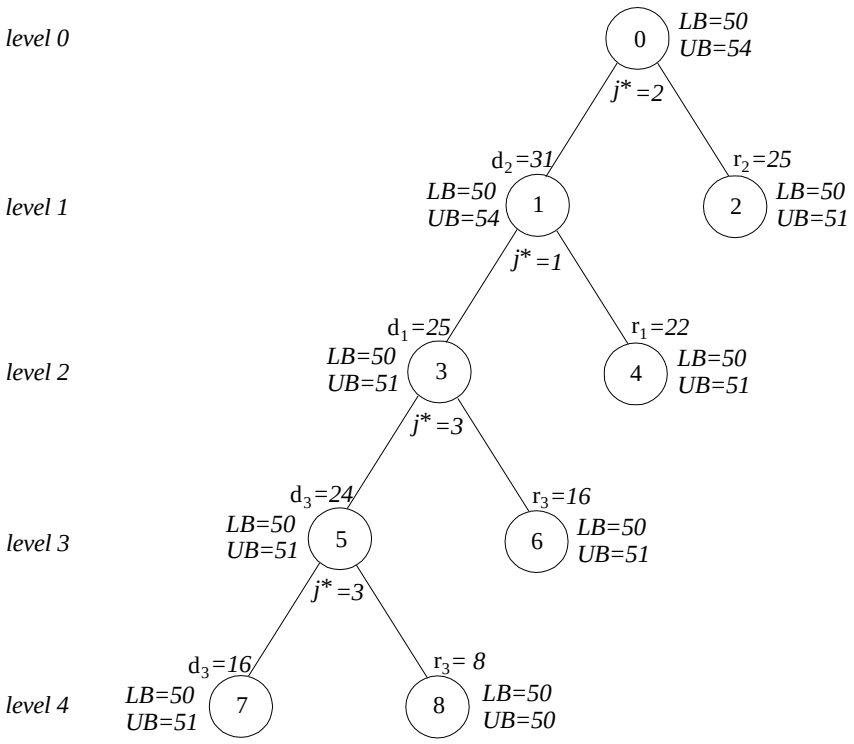


Figure 5.7. Example of Carlier's algorithm

We describe below the different operations performed for each node N of the tree.

$N = 0$ (root node) : $UB = UB(N) = 54$, $j_0 = 5$, $J_0 = \{5\}$, $LB_2(J_0) = 25$, $LB_0(N) = 50$, $LB(N) = 50$.

The set of deadlines is set equal to $\{46, 51, 43, 32, 37, 45, 45, 52, 30, 29\}$. The time interval $[5, 51]$ of job $j^* = 2$ (selected according to R_3) is

divided into the two ones $[5, 31]$ and $[25, 51]$. Nodes 1 and 2 are then created such that :

$UB(1) = 54$, $j_0 = 5$, $J_0 = \{5\}$, $LB_2(J_0) = 25$, $LB_0(1) = 50$,

$LB(1) = 50$.

$UB(2) = 51$, $j_0 = 5$, $J_0 = \{5\}$, $LB_2(J_0) = 25$, $LB_0(2) = 50$,

$LB(2) = 50$.

UB is then updated to 51.

$N = 1$: The deadlines are updated to $\{43, 31, 40, 29, 34, 42, 42, 49, 27, 26\}$. After adjusting $q_2 = 19$, we obtain : $UB(N) = 53$, $j_0 = 4$, $J_0 = \{4\}$, $LB_2(J_0) = 25$, $LB_0(N) = 50$, $LB(N) = 50$.

The time interval $[4, 43]$ of job $j^* = 1$ (selected according to R_2) is divided into the two ones $[4, 25]$ and $[22, 43]$. Nodes 3 and 4 are then created such that :

$UB(3) = 53$, $j_0 = 4$, $J_0 = \{4\}$, $LB_2(J_0) = 25$, $LB_0(3) = 50$,

$LB(3) = 50$.

$UB(4) = 51$, $j_0 = 5$, $J_0 = \{5\}$, $LB_2(J_0) = 25$, $LB_0(4) = 50$,

$LB(4) = 50$.

$N = 3$: After adjusting $q_1 = 25$, we obtain : $UB(N) = 54$, $j_0 = 5$, $J_0 = \{5\}$, $LB_2(J_0) = 25$, $LB_0(N) = 50$, $LB(N) = 50$.

The time interval $[1, 40]$ of job $j^* = 3$ (selected according to R_3) is divided into the two ones $[1, 24]$ and $[16, 40]$. Nodes 5 and 6 are then created such that :

$UB(5) = 54, j_0 = 5, J_0 = \{5\}, LB_2(J_0) = 25, LB_0(5) = 50$,

$LB(5) = 50$.

$UB(6) = 52$, $j_0 = 5$, $J_0 = \{5\}$, $LB_2(J_0) = 25$, $LB_0(6) = 50$,

$LB(6) = 50$.

$N = 5$: After adjusting $q_3 = 26$, we obtain : $UB(N) = 54$, $j_0 = 5$,

$J_0 = \{1, 2, 3, 4, 5, 9, 10\}$, $LB_2(J_0) = 47$, $LB_0(N) = 50$, $LB(N) = 50$.

The time interval $[1, 24]$ of job $j^* = 3$ (selected according to R_1) is divided into the two ones $[1, 16]$ and $[8, 24]$. Nodes 7 and 8 are then created such that :

$UB(7) = 54$, $j_0 = 5$, $J_0 = \{1, 2, 3, 4, 5, 9, 10\}$, $LB_2(J_0) = 47$,

$LB_0(7) = 50$, $LB(7) = 50$.

$UB(8) = 50$, $j_0 = 5$, $J_0 = \{1, 2, 3, 4, 5, 9, 10\}$, $LB_2(J_0) = 48$,

$LB_0(8) = 50, LB(8) = 50$.

UB is then updated to 50, which is the optimal makespan since the minimal lower bound in the tree is equal to 50. Figure 5.8 shows the optimal schedule of this instance.

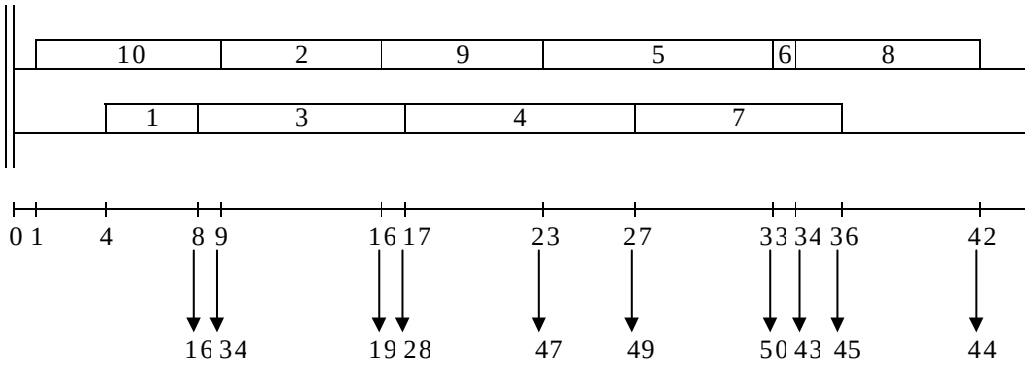


Figure 5.8. Optimal schedule of example 5.3

5.4 Adjustments and feasibility tests

Assume that we are given a $P|r_j, q_j|C_{\max}$ instance and denote by LB and UB a lower and an upper bound on its optimal solution, respectively. Let $\bar{J}$ be the set of jobs obtained after performing the preprocessing algorithm. Assume that the optimal makespan is strictly less than UB, then no job j can finish processing after $UB - q_j - 1$. This can be ensured by associating a deadline $d_j = UB - q_j - 1$ with each job $j \in \bar{J}$.

In this section, we develop several rules which provide sufficient conditions to prove that there is no feasible schedule with makespan less than UB. The importance of these rules is twofold. They are used in the branch-and-bound algorithm for discarding infeasible nodes, and they permit the adjustment of the heads and tails, so that the lower bounds are tightened. The proposed rules are based on the observation that if a job j is such that $d_j - r_j < 2p_j$,

then in any nonpreemptive schedule, there is necessarily one machine which has to process job j during the interval $[d_j - p_j; r_j + p_j]$ (cf. page 49). This provides a simple way to compute a lower bound on the number of machines which are necessarily loaded at any time. Therefore, the following feasibility condition holds.

Condition 5.1 : *The instance is infeasible if there is a time* $t \in [0, \max_{j \in \bar{J}} d_j]$ *such that the number of machines loaded at* t *is strictly greater than* m.

We recall that each job $j \in S = \left\{ j \in \bar{J}; d_j - r_j < 2p_j \right\}$, has a fixed processing part of $2p_j - (d_j - r_j)$ units which has to be processed in $[d_j - p_j, r_j + p_j]$, and each job $j \in \bar{J}$ has a free processing part $p'_j = \min(p_j, d_j - r_j - p_j)$ units which has to be processed in $[r_j, d_j - p_j] \cup [r_j + p_j, d_j]$. In addition to the notations provided in Section 4.2 (page 49), we denote by J_k the set of jobs which free parts may be processed during the time interval $E_k = [e_k, e_{k+1}]$ $(k = 1, ..., K-1)$, and by n_k the number of jobs in J_k.

Clearly, the amount of work in E_k $(k = 1, ..., K-1)$ cannot exceed

$$A_k = \min \left\{ \sum_{j \in J_k} p'_j \ ; \ (e_{k+1} - e_k) \times \min(n_k, m_k) \right\}$$

The infeasibility holds if the total time needed to process all the free parts exceeds the total amount of work possible in the intervals. This leads to the following feasibility condition.

Condition 5.2 : *The instance is infeasible if* $\sum_{k=1}^{K-1} A_k < \sum_{j \in \bar{J}} p'_j$.

A *time window* is defined as an interval of time $[a_h, b_h] \subseteq [0, UB]$, in which there is at least one idle machine. Let W denote the set of the different time windows, and let $w = |W|$. Provided the number of loaded machines at any time, the set W can be easily derived. These time windows will be used in order to adjust heads and tails of any job $j_0 \in \bar{J} \backslash S$. The job j_0 can start processing at r_{j_0} in a feasible schedule if there exists a time window $[a_h, b_h]$ $(h = 1, ..., w)$ such that $[r_{j_0}, r_{j_0} + p_{j_0}] \subseteq [a_h, b_h]$. That is, $a_h \leq r_{j_0} < r_{j_0} + p_{j_0} \leq b_h$. The earliest starting time of job $j_0 \in \bar{J} \backslash S$ may be computed by the following algorithm.

Adjusting heads

Step 1. *Find the maximal* h $(1 \leq h \leq w)$ *such that* $a_h \leq r_{j_0}$.

Step 2. *While* $(r_{j_0} + p_{j_0} > b_h)$, *Begin Set* $h = h+1$, $r_{j_0} = a_h$, *End (While).*

Similarly, job $j_0 \in \bar{J} \backslash S$ can finish processing at d_{j_0} in a feasible schedule if there exists h $(h = 1, ..., w)$ such that $a_h \leq d_{j_0} - p_{j_0} < d_{j_0} \leq b_h$. In this case also, it is possible to adjust d_{j_0} to the latest possible finishing time using a similar procedure.

Adjusting deadlines and tails

Step 1. *Find the minimal* h $(1 \leq h \leq w)$ *such that* $b_h \geq d_{j_0}$.

Step 2. *While* $(d_{j_0} - p_{j_0} < a_h)$, *Begin Set* $h = h - 1$, $d_{j_0} = b_h$, $q_{j_0} = \max(q_{j_0}, LB - d_{j_0})$ *End (While).*

Clearly, if $w = 1$, then the above adjustments are not possible. It is worth noting that the heads and tails of a job $j_0 \in S$ can also be adjusted using a similar approach. However, no adjustment holds if $|S| < m$. Note that a job $j_0 \in S$ such that $r_{j_0} + p_{j_0} = d_{j_0}$ has a fixed starting time r_{j_0} and a fixed finishing time d_{j_0} in any feasible schedule. Thus, no adjustments have to be performed on such a job. After performing these adjustments a third feasibility condition is the following :

Condition 5.3 : *If there is a job* $j_0 \in \bar{J}$ *such that* $r_{j_0} + p_{j_0} > d_{j_0}$, *then the instance is infeasible.*

The algorithm described below and denoted by *Feasibility and Adjustment Procedure (FAP)* is designed in order to check the feasibility of any instance and to adjust the heads and tails of the jobs.

FAP description

Step 1. *Determine the set* S. *If* $|S| < m$, *then stop.*

Step 2. *If Condition 5.1 is satisfied, then the instance is infeasible. Stop.*

Step 3. *If Condition 5.2 is satisfied, then the instance is infeasible. Stop.*

Step 4. *Determine the set* W *of time windows. If* $w = 1$, *then stop.*

Step 5. *Adjust the head, deadline and tail of every job* $j \in \bar{J}$.

Step 6. *If Condition 5.3 is satisfied, then the instance is infeasible. Stop.*

Step 7. *If there is no adjustment, then stop. Else, update* $\bar{J}$ *and go to step 1.*

Example 5.4 : Consider the 6 job-3 machine instance defined by Table 5.6. Assume that we are given $LB = 19$ and $UB = 21$. The deadlines are set to:

$$d_1 = 13,\ d_2 = 14,\ d_3 = 15,\ d_4 = 10,\ d_5 = 12,\ d_6 = 16$$

At the first iteration, we have $\bar{J} = J$ and $S = \{2, 5, 6\}$. Using Observation 4.2, there must exist one machine loaded with job 2 during the interval $[5, 12]$, a second one loaded with job 5 during the interval $[3, 11]$, and a third one loaded with job 6 during the interval $[9, 15]$. Hence, the three machines are necessarily loaded during the interval $[9, 11]$. Therefore, we have $W = \{[0, 9]\,; [11, 21]\}$.

Note that job 1 is released at $r_1 = 10$ but cannot start processing before time 11. Consequently, its release date is adjusted to $r_1 = 11$. Moreover, job 3 has a deadline equal to $d_3 = 15$ but cannot be processed in the time window $[11, 21]$ since $d_3 - p_3 = 8 < 11$. Therefore, its deadline and tail are adjusted to $d_3 = 9$ and $q_3 = 10$, respectively. In a similar way, the deadline of job 4 is adjusted to $d_4 = 9$.

At the second iteration, we have $\bar{J} = \{2, 3, 4, 5, 6\}$, $J_Q = \{1\}$ and $S = \{2, 3, 4, 5, 6\}$. Using Observation 4.2, there must exist one machine loaded with job 2 during the interval $[5, 12]$, a second one loaded with job 3 during the interval $[2, 8]$, a third one loaded with job 4 during the interval $[7, 8]$ and a fourth one loaded with job 5 during the interval $[3, 11]$. Therefore, there must exist *four* machines loaded during the interval $[7, 8]$. Consequently, the instance is identified to be infeasible (cf. Condition 5.1).

j	1	2	3	4	5	6
r_j	10	3	1	6	2	8
p_j	1	9	7	2	9	7
q_j	7	6	5	10	8	4

Table 5.6. Data of the 6 job - 3 machine instance of example 5.4

Clearly, the adjustment of heads and tails may improve the tightness of the bounds. In particular, the semi-preemptive lower bound can be tightened in the following way. For each trial value, the FAP is first applied. If the instance is infeasible, then we move to the next trial value. Otherwise, the max-flow network is constructed using the adjusted heads and deadlines. Moreover, after applying the FAP, it is possible to improve a lower bound in the

following way. Initially, the lower bound is set to $LB = \max\{LB, LB_0(J)\}$. If the FAP proves that there is no feasible schedule with makespan less than or equal to a trial value $C \in [LB, UB]$, then the lower bound is updated to $C + 1$. Therefore, a FAP based lower bound, denoted FAP_LB, may be obtained by performing a bisection search on the interval $[LB, UB]$. For that purpose, we propose the following algorithm. The interval $[C^-, C^+]$ denotes the current search interval of C.

Computation of FAP_LB

Step 0. *Set* $C^- = \max\{LB, LB_0(J)\}$, $C^+ = UB$, *and* $FAP\_LB = C^-$.
Step 1. *Set* $C = \left\lfloor \dfrac{C^- + C^+}{2} \right\rfloor$.
Step 2. *Set* $d_j = \min\{d_j, C - q_j\}$ *and* $q_j = \max\{q_j, FAP\_LB - d_j\}$ *for all* $j \in \bar{J}$.
Step 3. *Apply FAP to the instance obtained in step 2. If the instance is infeasible, then set* $C^- = C + 1$ *and* $FAP\_LB = C^-$. *Else, set* $C^+ = C$.
Step 4. *If* $C^- = C^+$, *then stop. Else, go to step 1.*

5.5 Synthesis of the new branch-and-bound algorithm

In this section, we describe how the different contributions developed in this thesis are used in a time-windows-based branch-and-bound algorithm.

We implemented our algorithm using at each node N, the lower bound:

$$LB(N) = \max\{LB_0(J), LB^*(\overline{J}(N))\}$$

Obviously, $LB_0(J)$ is computed first, and $LB^*(\overline{J}(N))$ is computed only if $LB_0(J) < UB$. Otherwise, the node is pruned.

The value of the upper bound computed for the node N is :

$$UB(N) = \min\left\{JS(J), JS(\overline{J}(N)), JS^{-1}(J), JS^{-1}(\overline{J}(N))\right\}$$

The branching rule is similar to that proposed by Carlier (1987) (cf. page 92). However, we implemented a slightly different criterion for selecting the job to be branched. Job j^* is selected according to a slight modification of rule R_2. The schedule considered in R_2 is that of the minimal makespan among the four ones constructed for node N. The selected job $j^* \in \overline{J}(N)$

is the one with smallest tail among those on the critical path and with tail smaller than q_{j_0}, where j_0 is the critical job with maximal tail. If there is no job satisfying this condition, then j^* is selected according to R_3.

Before computing the lower bounds for N_1 and N_2, one has to perform some preliminary operations. First, the preprocessing algorithm is applied to node N_1(respectively N_2). Then FAP is applied to N_1(respectively N_2) and FAP_LB is computed. In case of feasibility, the upper bound $UB(N_1)$ (respectively $UB(N_2)$) is computed and UB is updated. Finally, the lower bound is computed for a node N only if it was not proven infeasible and $\left|\overline{J}(N)\right| > m$.

A pseudo-code description of our branch-and-bound algorithm is the following:

Step 0. Initialization

0.1. *Make a root node R containing the data set of the problem.*

0.2. *Apply the preprocessing algorithm to node R.*

0.3. *Compute $UB(R)$. Set $UB = UB(R)$.*

0.4. *If $\left|\bar{J}(R)\right| \leq m$, then $C^*_{\max} = UB$. Stop. Else, compute $LB(R)$ and set $\Sigma = \{R\}$ (Σ : is the set of candidate nodes).*

Step 1. Termination test, Node selection and Adjustment

1.1. *If $\Sigma = \emptyset$, then go to step 4. Else, Select a node $N \in \Sigma$ with minimal lower bound.*

1.2. *If $LB(N) \geqslant UB$, then go to step 4.*

1.3. *Adjust $q_j(N)$ and $d_j(N)$ using equations (5.1) and (5.2) for all $j \in \overline{J}(N)$. If a margin is negative, then prune node N and go to step 1.1.*

Step 2. Preprocessing and FAP

2.1. *Apply preprocessing algorithm to node N.*

2.2. *Apply FAP to node N, and compute $FAP\_LB(N)$. If N is feasible, $\left|\overline{J}(N)\right| > m$ and $FAP\_LB(N) < UB$ then go to step 3. Else, prune N and go to step 1.*

Step 3. Branching

3.1. *Select* j^* *and create nodes* N_1 *and* N_2. *Set* $\Sigma = \Sigma \cup \{N_1, N_2\}$.

3.2. *For* i = 1,2 *do*

3.2.1. *Apply preprocessing to* N_i *and compute* $UB(N_i)$.

3.2.2. *If* $UB(N_i) < UB$, *then set* $UB = UB(N_i)$.

3.2.3. *If* $\left|\bar{J}(N_i)\right| \leq m$, *then prune node* N_i *and set* $\Sigma = \Sigma \backslash \{N_i\}$.

3.2.4. *Compute LB(*N_i*). If* $LB(N_i) \geq UB$, *then prune* N_i *and set* $\Sigma = \Sigma \backslash \{N_i\}$.

3.2.5. *Apply FAP to* N_i *and compute* $FAP\_LB(N_i)$. *If* $FAP\_LB(N_i) \geq UB$, *then prune* N_i *and set* $\Sigma = \Sigma \backslash \{N_i\}$.

3.3. *Prune N, set* $\Sigma = \Sigma \backslash \{N\}$ *and go to step 1.*

Step 4. Optimal makespan *Set* $C^*_{\max} = UB$. *Stop.*

5.6 A Forward-Backward implementation

In order to take advantage of the symmetry of the $P|r_j, q_j|C_{\max}$, the Forward and the Backward problems are solved in parallel. In this way, the convergence of the algorithm may be accelerated through an exchange of information between the two problems. In this section, we propose a *pseudo-parallel* implementation of a branch-and-bound algorithm, hereafter called the *Forward-Backward branch-and-bound algorithm*. Two search trees are constructed in a way similar to that described in the previous section. We alternate the exploration of both trees in the following way. Assume that at an iteration, a node from the Forward (Backward) tree is explored, then at the next iteration a node from the Backward (Forward) tree is explored. The upper bound UB is taken as the minimal value of $UB(N)$ over all the nodes N of both trees. At each iteration, the lower bound $LB(N)$ of a node N in a given tree is updated to $\max\{LB(N), LB_{\min}\}$, where $LB_{\min}$ is the minimal lower bound over all the nodes of the other tree. The process continues until a schedule is proved optimal.

5.7 Computational experiments

In this section we present the results of an analysis of the performance of our new branch-and-bound algorithm as well as Carlier's algorithm. The following notation is adopted :

- $B\&B1$: the proposed time-windows-based algorithm
- CAR : the algorithm presented by Carlier (1987)

The two algorithms are coded in C and compiled with Visual C++ 5.0. All the computational experiments were carried out on a Pentium IV 2 GHz Personal Computer with 256 MB RAM.

5.7.1 Test problems

The test problems are generated in a similar way as in Carlier (1987). The number of jobs n is taken equal to 50, 100, 150, 200, 250 and 300 jobs. The number of machines m is taken equal to 2, 3, and 4 machines. The processing times are drawn from the discrete uniform distribution on [1,10]. The heads and tails are drawn from the discrete uniform distribution on $[1, K\frac{n}{m}]$, where K is a positive integer set equal to 1, 3, 5, and 7. We combined these problem characteristics to obtain 72 classes of randomly generated test problems. For each class, 10 instances are generated which results in 720 test problems. A CPU time limit of 300 sec. is fixed for each run.

5.7.2 Performance of the algorithms

The results of our analysis are reported in Tables 5.7, 5.8 and 5.9. For each algorithm, we provide:

- $Time$: mean CPU time (in sec.)
- NN_{mean} : mean number of nodes
- $NN_{\max}$: maximal number of nodes
- US : percentage of instances for which optimality was not proved after reaching the time limit

- Gap_{mean} : average gap of unsolved instances, where the gap is defined as the ratio
$$100(\frac{UB - LB}{LB})$$
- $Gap_{\max}$: maximal gap of unsolved instances

Global performance analysis

Table 5.7 provides strong evidence that the algorithm developed in this chapter consistently outperforms Carlier's algorithm (CAR). We observe that CAR solved only 76.66% of the instances within the time limit of 300 sec., while $B\&B1$ solved 93.89% of the instances within this time limit. The mean CPU time of $B\&B1$ is 20.04 sec., and this algorithm is 3.67 times faster than CAR.

The difference between our algorithm and CAR is even more dramatic when the mean number of explored nodes criterion is considered. Indeed, CAR explores on average 52.35 times more nodes than $B\&B1$ does.

	$Time$	NN_{mean}	$NN_{\max}$	US	Gap_{mean}	$Gap_{\max}$
$B\&B1$	20.04	963.45	48632	6.11	0.68	1.51
CAR	73.59	50443.89	2721026	24.44	0.69	4.93

Table 5.7. Performance of the algorithms

Analysis of the performance with respect to n

The results presented in Table 5.8 show that large instances with up to 300 jobs are solved within a very moderate CPU time. For instance, 95% of the 300-job instances were solved to optimality by $B\&B1$ with a mean CPU time of 19.65 sec. The maximal gap of the unsolved instances is 0.24%.

Table 5.8 shows that for both algorithms the mean (maximal) number of explored nodes and the mean (maximal) gap tend to decrease as n increases. However, the CPU time and the percentage of unsolved instances (US) exhibit a more complex (and correlated) behavior. Indeed, we observe that for our algorithm, the mean computing time decreases when n varies from 50 to 150 and then increases when n varies from 150 to 300. This may be explained by noting that the total computation time is roughly equal to the product of the total number of explored nodes by the computing time required by each node. As it has been noticed above, the total number of explored nodes

decreases as n increases. However, the computational burden at each node increases as n increases and this trend becomes dominant for larger values of n.

	n	$Time$	NN_{mean}	$NN_{\max}$	US	Gap_{mean}	$Gap_{\max}$
	50	37.65	4846.76	48632	12.50	1.17	1.51
	100	22.70	578.89	11216	7.50	0.72	1.29
	150	7.88	85.38	3243	2.50	0.46	0.48
$B\&B1$	200	13.94	100.33	2198	4.16	0.35	0.38
	250	18.41	103.91	1785	5.00	0.27	0.30
	300	19.65	65.40	1382	5.00	0.20	0.24
	50	110.13	268495.03	2721026	36.66	1.48	4.93
	100	80.08	24941.91	539137	26.66	0.71	2.01
	150	57.56	4613.24	132917	19.16	0.47	1.10
CAR	200	75.15	3248.84	69116	25.00	0.33	1.06
	250	55.32	960.98	15949	18.33	0.27	0.55
	300	63.33	403.35	3738	20.83	0.25	0.95

Table 5.8. Performance of the algorithms for each problem size

Analysis of the performance with respect to m and K

Table 5.9 depicts the behavior of $B\&B1$ when the number of machines m and the value of K vary. It is worth noting that Carlier's algorithm exhibits a similar behavior. The value of K seems to be the factor which most determines the difficulty of the problems. This result is consistent with the one observed by Carlier (1987). When $K \in \{5, 7\}$ (i.e., instances with large heads and tails) the problem can be considered as "easy". We observe that for this problem class all the 360 instances are solved in a negligible CPU time. In contrast, it appears that problems with $K = 1$ and 3 are much harder to solve, and this difficulty strongly increases when m increases. The hardest problem class, for which 40% of the instances remained unsolved after reaching the time limit, is obtained with $m = 4$ and $K = 1$. This (relatively) modest performance may be explained by the fact that when $K\frac{n}{m}$ is small, then the heads and tails are also small. Therefore, very few jobs may satisfy conditions (3.1) and (3.2), which implies that the preprocessing algorithm will fail to reduce the jobset significantly. It is worth noting that $B\&B1$ provided a mean gap of 0.64% for these hard instances.

		$Time$	NN_{mean}	$NN_{\max}$	US	Gap_{mean}	$Gap_{\max}$
$m = 2$	$K = 1$	2.31	16.90	227	0.00	-	-
	$K = 3$	20.82	1996.43	41565	6.66	0.82	1.49
	$K = 5$	0.92	3.51	22	0.00	-	-
	$K = 7$	0.43	1.68	19	0.00	-	-
$m = 3$	$K = 1$	26.40	771.01	39418	6.66	0.43	1.16
	$K = 3$	31.34	2347.75	36867	10.00	0.79	1.05
	$K = 5$	0.74	3.21	40	0.00	-	-
	$K = 7$	0.26	1.23	10	0.00	-	-
$m = 4$	$K = 1$	125.42	4686.85	44972	40.00	0.64	1.51
	$K = 3$	31.15	1729.51	48632	10.00	0.84	1.42
	$K = 5$	0.39	1.98	20	0.00	-	-
	$K = 7$	0.27	1.30	17	0.00	-	-

Table 5.9. Sensitivity of $B\&B1$ to the variation of m and K

5.7.3 Impact of the different components

Since it is well known that the performance of branch-and-bound algorithms strongly relies on the quality of the implemented bounds, one may legitimately wonder whether an implementation of improved lower and upper bounds suffices to make CAR competitive with $B\&B1$. In order to investigate this issue, we embedded within CAR the lower and/or upper bounds developed in this thesis, and we compared the performance of the resulting algorithm with $B\&B1$. Table 5.10 depicts the results of the experiments. In this table, each entry represents the ratio of the measure obtained with the "improved" CAR with respect to $B\&B1$. We observe that although the new lower and/or upper bounds significantly improve the performance of CAR, the resulting improved variants of CAR are still lagged far behind $B\&B1$. Indeed, embedding the lower bound LB^* together with the improved upper bound yields an algorithm which requires, on average, 74% more CPU time and explores 13% more nodes than $B\&B1$ does.

	$Time_{ratio}$	$Mean\ NN_{ratio}$	US_{ratio}
CAR	3.67	52.35	4.00
$CAR + LB^*$	2.94	1.90	3.00
$CAR + UB$	2.59	10.69	2.81
$CAR + LB^* + UB$	1.74	1.13	1.79

Table 5.10. Relative performance of the bounds with respect to $B\&B1$

Pushing the investigation a step further, we have evaluated the pertinence of each component (lower/upper bounds, preprocessing, FAP, Forward-Backward strategy). For that purpose, we have implemented five variants of our algorithm. These variants are the following:

- $B\&B1\backslash LB$: The lower bound LB^* has been replaced by Carlier's lower bound CLB
- $B\&B1\backslash UB$: Only one upper bound is computed for each node. This bound corresponds to the makespan of Jackson's schedule constructed for the entire jobset J
- $B\&B1\backslash PREP$: The preprocessing algorithm is not implemented
- $B\&B1\backslash FAP$: The FAP is not implemented
- $B\&B1\backslash FB$: There is no Forward-Backward implementation

We compared each of these variants to $B\&B1$. The results are displayed in Table 5.11. We adopted the following notation:

$Time_{inc}$: the percentage increase of the mean CPU time
$Mean\ NN_{inc}$: the percentage increase of the mean number of nodes
US_{inc} : the percentage increase of the number of unsolved instances

Table 5.11 shows the worth of implementing each of the proposed components since $B\&B1$ consistently outperforms all the five variants. Indeed, we observe that $B\&B1\backslash UB$ and $B\&B1\backslash FB$ yield the two most important increases of the mean CPU time (90.26% and 52.29%, respectively). It is worth to note that a very similar behavior is observed for the percentage of unsolved instances. Moreover, $B\&B1\backslash LB$ exhibit a dramatic increase in the mean number of explored nodes (730.43%). Furthermore, skipping the preprocessing phase increases the mean CPU time by 18.16% and the mean number of explored nodes by 22.17%. We also observe that the impact of implementing FAP is more important when the mean number of explored nodes is to be considered.

	$Time_{inc}$	$Mean\ NN_{inc}$	US_{inc}
$B\&B1\backslash LB$	16.66	730.43	24.87
$B\&B1\backslash UB$	90.26	35.94	93.12
$B\&B1\backslash PREP$	18.16	22.17	20.45
$B\&B1\backslash FAP$	4.29	30.87	4.41
$B\&B1\backslash FB$	52.29	37.69	61.37

Table 9. Impact of different components on $B\&B1$

Chapter 6

A sequence-based branch-and-bound algorithm for the $P|r_j, q_j|C_{\max}$

In this chapter, we propose a second exact algorithm for the $P|r_j, q_j|C_{\max}$. This branch-and-bound algorithm is based on formulating the solution schedule as a sequence of jobs. Extensive computational experiments show that this algorithm makes it feasible to solve exactly very large instances with up to 1500 jobs and 50 machines in a small CPU time.

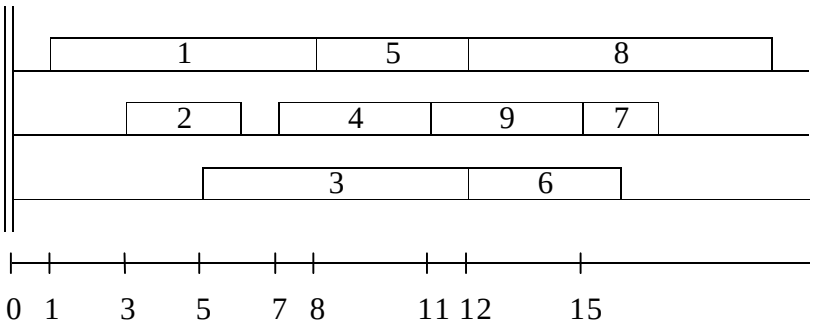
6.1 A sequence representation of $P|r_j, q_j|C_{\max}$ solutions

The fundamental idea of our algorithm is based on the fact that each schedule $S = (t_1, t_2, ..., t_n)$ can be coded as a sequence $\sigma = (\sigma(1), \sigma(2), ..., \sigma(n))$ of the n jobs (t_j denotes the starting time of job j). This sequence is simply obtained by sequencing the jobs in S according to the nondecreasing order of their starting times. Formally, a schedule S can be coded as a sequence σ if and only if the following property is satisfied:

$$t_{\sigma(k)} \leq t_{\sigma(k+1)} \text{ for all } k = 1, ..., n-1$$

This coding procedure is illustrated by the following example.

Example 6.1. Consider the schedule depicted in Figure 6.1. This schedule can be coded by the sequence $\sigma = (1, 2, 3, 4, 5, 9, 8, 6, 7)$.



$$\sigma = (1, 2, 3, 4, 5, 9, 8, 6, 7)$$

Figure 6.1. A feasible schedule and its corresponding sequence

Clearly, several schedules can be coded by the same sequence. Assume that we are given a sequence $\sigma = (\sigma(1), \sigma(2), ..., \sigma(n))$. The procedure described below decodes the sequence σ in order to construct a feasible corresponding schedule S_σ. The availability of machine M_i is denoted by u_i $(i = 1, ..., m)$. The first available machine is denoted by M_0 and its availability is denoted by u_0.

Decode(σ)
Set $u_i = 0$ *for* $i = 1, ..., m$
For all $k = 1, ..., n$ *do*
Schedule job $\sigma(k)$ *on* M_0. *Set* $t_{\sigma(k)} = r_{\sigma(k)}$ *and* $u_0 = t_{\sigma(k)} + p_{\sigma(k)}$
Update M_0 *and* u_0
Set $r_{\sigma(h)} = \max(r_{\sigma(h)}, t_{\sigma(k)}, u_0)$ *for all* $h = k + 1, ..., n$
End (do)

The following example illustrates this decoding procedure.

Example 6.2. Consider the 5 job - 2 machine instance defined by Table 6.1. Let $\sigma = (3, 1, 4, 5, 2)$ denote a sequence of the jobs. The corresponding schedule S_σ is depicted in figure 6.2.

j	1	2	3	4	5
r_j	1	3	2	3	10
p_j	7	2	4	1	3
q_j	2	7	4	2	1

Table 6.1. The 5 job - 2 machine instance of example 6.2

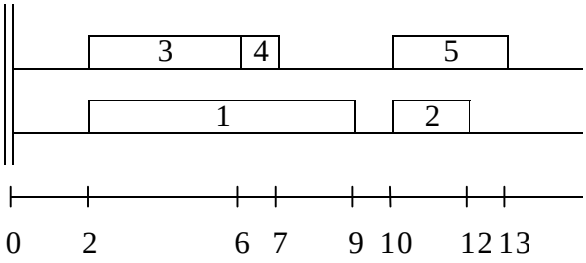


Figure 6.2. The decoded schedule of example 6.2

Lemma 6.1.

The schedule S_σ constructed by procedure **Decode**(σ) *is the schedule with minimal makespan among those which can be coded by σ*

Proof. Let S be any schedule which can be coded by σ. Let t_j and t_j^* denote the starting time of job j in S and S_σ, respectively. In order to prove that the makespan of S_σ is less than or equal to that of S, we will prove by induction that $t^*_{\sigma(k)} \leq t_{\sigma(k)}$ $(k = 1, ..., n)$.

Clearly, we have $t^*_{\sigma(1)} = r_{\sigma(1)} \leq t_{\sigma(1)}$. Now, assume that $t^*_{\sigma(k)} \leq t_{\sigma(k)}$ for a given k $(k = 1, ..., n-1)$ and let us prove that $t^*_{\sigma(k+1)} \leq t_{\sigma(k+1)}$.

We have $t^*_{\sigma(k+1)} = \max(r_{\sigma(k+1)}, t^*_{\sigma(k)}, u_0^k)$, where u_0^k denotes the availability of the first available machine in S_σ once job $\sigma(k)$ has been scheduled. The result clearly holds if $t^*_{\sigma(k+1)} = \max(r_{\sigma(k+1)}, t^*_{\sigma(k)})$ since $r_{\sigma(k+1)} \leq t_{\sigma(k+1)}$ and $t^*_{\sigma(k)} \leq t_{\sigma(k)} \leq t_{\sigma(k+1)}$.

Consider the case where we have $t^*_{\sigma(k+1)} = u_0^k > \max(r_{\sigma(k+1)}, t^*_{\sigma(k)})$. Let Θ denote the set of the m jobs processed on the machines between $t^*_{\sigma(k)}$ and u_0^k in the schedule S_σ. Since these jobs precede $\sigma(k)$ in σ, then we have

$$t_j^* \leq t_j \leq t_{\sigma(k)} \text{ for all } j \in \Theta$$

Hence, all of the m jobs of Θ start processing before $t_{\sigma(k)}$ in S.

On the other hand, we have $t_j^* + p_j \leq t_j + p_j$ for all $j \in \Theta$. Therefore, $u_0^k = \min_{j\in\Theta} (t_j^* + p_j) \leq \min_{j\in\Theta} (t_j + p_j)$. That is, all of the m jobs of Θ finish processing after u_0^k in S.

Consequently, the m jobs of Θ are being processed during the interval $[t_{\sigma(k)}, u_0^k]$ in S. Since $t_{\sigma(k+1)} \geq t_{\sigma(k)}$, then the job $\sigma(k+1)$ cannot start processing before u_0^k in S. ■

An immediate consequence of Lemma 6.1 is that the problem turns to find the optimal sequence instead of finding the optimal schedule.

6.2 Data representation

For each node N of the search tree, we associate a partial sequence σ_N of scheduled jobs. It means that node N represents all the sequences beginning with σ_N. Obviously, assuming that there are some scheduled jobs means that the machines have different availability times. Let $\bar{J}(N)$ denote the set of unscheduled jobs. Then, the subproblem defined on $\bar{J}(N)$ is a parallel machine problem with heads, tails and machine availabilities $P, NC_{inc}|r_j, q_j|C_{\max}$. Note that the machine availability times can be easily deduced by decoding the partial sequence σ_N. Let $u_0(N)$ denote the availability time of the first available machine.

A lower bound $LB(N)$, an upper bound $UB(N)$ and deadlines $d_j(N)$ ($j \in \bar{J}(N)$) are associated to the node N. If UB denotes the current best upper bound, then the deadlines are initially set to $UB - q_j(N) - 1$. Moreover, a starting time $t_j(N)$ is defined for $j \in \sigma_N$.

6.3 Branching scheme

The purpose of the branching rule is to indicate a candidate job to be appended on the first available position of the sequence σ_N, and to generate a descendant of the current node in the search tree. Given a node N_0, if a job $j_0 \in \overline{J}(N_0)$ is to be appended to σ_{N_0}, then a descendant node N of N_0 is created with the following data:

Machine data

- $u_i(N) = u_i(N_0)$ for $i = 1, ..., m$
- $u_0(N) = r_{j_0}(N_0) + p_{j_0}$

- $u_0(N) = \min_{i=1,\ldots,m} u_i(N)$

Data of scheduled jobs

- $\sigma_N = \sigma_{N_0} j_0$
- $t_{j_0}(N) = r_{j_0}(N_0)$
- $d_{j_0}(N) = r_{j_0}(N_0) + p_{j_0}$
- $q_{j_0}(N) = \max(q_{j_0}(N), LB(N_0) - d_{j_0}(N))$

Data of unscheduled jobs

- $\bar{J}(N) = \bar{J}(N_0) \backslash \{j_0\}$
- $r_j(N) = \max(r_j(N_0), t_{j_0}(N), u_0(N))$ for all $j \in \bar{J}(N)$
- $q_j(N) = q_j(N_0)$ for all $j \in \bar{J}(N)$
- $d_j(N) = d_j(N_0)$ for all $j \in \bar{J}(N)$

During the computations, whenever an improved upper bound UB is found, the deadline and the tail of all the jobs $j \in \bar{J}(N)$ are adjusted to

$$d_j(N) = \min \{d_j(N), UB - q_j(N) - 1\}$$

$$q_j(N) = \max \{q_j(N), LB(N) - d_j(N)\}$$

The depth-first search strategy has been adopted. It consists in branching on the first candidate node descendant of the current node in the tree. With no loss of generality, the jobs of $\bar{J}(N)$ are ranked according to nondecreasing release dates, and in case of ties, nonincreasing delivery times and nondecreasing processing times.

For the sake of clarity, we will denote the partial sequence σ_N by σ, the set $\bar{J}(N)$ by $\bar{J}$, and so on.

6.4 Node selection rules

In this section, we derive immediate selection rules which aims at removing dominated nodes from the set of candidate nodes to be branched.

Let $C^*_{\max}(\sigma)$ denote the minimum makespan of all those of the sequences beginning with the partial sequence σ. The three following results will be used to derive dominance relations between jobs of $\bar{J}$ to be appended to σ.

Observation 6.1.

Let j and j' be two jobs of $\bar{J}$ such that

$$r_j = r_{j'}, p_j = p_{j'} \text{ and } q_j = q_{j'}$$

Then,

$$C^*_{\max}(\sigma j) = C^*_{\max}(\sigma j')$$

Proof. Obvious. ■

Observation 6.2.

Let $j_0 \in \bar{J}$ denote the job such that $r_{j_0} + p_{j_0} = \min_{j \in \bar{J}} (r_j + p_j)$. Assume that there exists a job $j \in \bar{J}$ such that

$$r_j \geq r_{j_0} + p_{j_0}$$

Then,

$$C^*_{\max}(\sigma j_0 j) \leq C^*_{\max}(\sigma j)$$

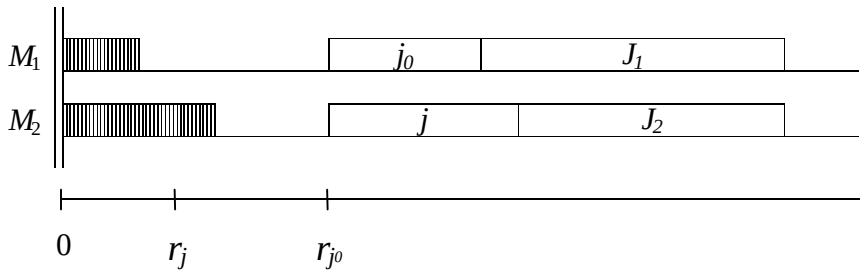
Proof. Consider any sequence beginning with the partial sequence σj. Sequencing job j_0 between σ and j will diminish the starting time of j_0 without delaying the starting times of the other jobs. ■

Observation 6.3.

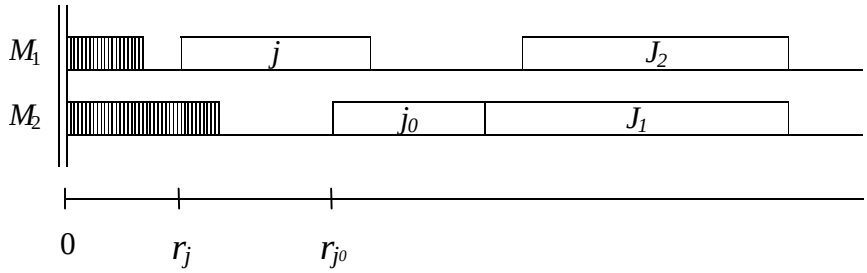
Let $\bar{u}_i$ denote the availability time of the i^{th} available machine $(i = 1, ..., m)$. Assume that there is a job j_0 such that $r_{j_0} \geq \bar{u}_2$. Let j denote a job such that $r_j \leq r_{j_0}$. Then,

$$C^*_{\max}(\sigma j j_0) \leq C^*_{\max}(\sigma j_0 j)$$

Proof. Let M_1 and M_2 denote the first and second available machines. In any sequence beginning with the partial sequence $\sigma j_0 j$, the job j_0 is scheduled at $t_{j_0} = r_{j_0}$ on M_1 and the job j is scheduled at $t_{j_0} = \max(r_j, t_{j_0}, \bar{u}_2)$ on M_2 (cf. figure 6.3a). Let J_1 denote the set containing j_0 and all jobs that are processed after j_0 on M_1, and J_2 denote the set containing job j and all jobs processed after j on M_2. Since all jobs in J_1 and J_2 start processing after $t_{j_0} \geq \max(\bar{u}_1, \bar{u}_2)$, then the schedule obtained by interchanging J_1 and J_2 has the same makespan as the original one. This latter schedule is clearly dominated by the sequence beginning with $\sigma j j_0$ and depicted in figure 6.3b. ■



6.3a. A schedule beginning with $\sigma j_0 j$



6.3b. A schedule beginning with $\sigma j j_0$

Figure 6.3. $C^*_{\max}(\sigma j j_0) \leq C^*_{\max}(\sigma j_0 j)$

The following selection rules are immediate consequences of the above observations. The first two rules are derived from Observations 6.1 and 6.2, respectively, whereas the two last ones are derived from Observation 6.3. The jobs $j_1, j_2, ..., j_n$ denote the jobs of $\bar{J}$ sorted according to the nondecreasing order of their release dates.

Selection rules

$\mathbf{R}_1$: If two jobs j_h and j_k $(h < k)$ of $\bar{J}$ have equal heads, processing times and tails, then job j_k is not candidate to be appended to σ

$\mathbf{R}_2$: All jobs $j \in \bar{J}$ such that $r_j \geq \min_{j \in \bar{J}} (r_j + p_j)$ are not candidate to be appended to σ

$\mathbf{R}_3$: Assume that there is a job $j_k \in \bar{J}$ such that $r_{j_k} \geq \bar{u}_2$. Then, only jobs j_h $(h = k+1, ..., n)$ are candidate to be appended to σj_k

$\mathbf{R}_4$: If $r_{j_n} \geq \bar{u}_2$, then job j_n is not candidate to be appended to σ

The following example shows how these selection rules are used in reducing the number of nodes in the tree.

Example 6.3. Consider a given partial sequence σ in a two-machine instance. Let $\bar{J} = \{1, 2, 3, 4, 5\}$ which data are provided by Table 6.2. Assume that the machine availabilities are $\{2, 8\}$.

j	1	2	3	4	5
r_j	2	2	5	12	13
p_j	6	6	1	6	2
q_j	1	1	3	1	8

Table 6.2. Data of the unscheduled jobs of example 6.3

Figure 6.4 illustrates the significant impact of the proposed selection rules on the size of the tree. Indeed, applying the selection rules reduces the number of potentially valid nodes at the second level from 20 nodes to only 3 nodes. The details are provided in the following.

Clearly, nodes $\sigma 2$ and $\sigma 5$ are removed according to R_1 and R_4, respectively. Also, nodes $\sigma 41, \sigma 42$ and $\sigma 43$ are removed according to R_3.

Assume that job 1 is appended to σ. Then, the release dates corresponding to $\bar{J} = \{2, 3, 4, 5\}$ are $\{8, 8, 12, 13\}$. Therefore, according to R_2, the jobs with release dates larger than or equal to $\min_{j \in \bar{J}} (r_j + p_j) = 9$ are not candidate to be appended to $\sigma 1$. That is, nodes $\sigma 14$ and $\sigma 15$ are removed.

Similarly, assume that job 3 is appended to σ. Then, the release dates corresponding to $\bar{J} = \{1, 2, 4, 5\}$ are $\{6, 6, 12, 13\}$. Therefore, node $\sigma 32$ is

removed according to R_1, and nodes $\sigma 34$ and $\sigma 35$ are removed according to R_2.

Finally, in the case where job 4 is appended to σ, the release dates corresponding to $\bar{J} = \{1, 2, 3, 5\}$ are $\{12, 12, 12, 13\}$. Since $r_5 = \min_{j \in \bar{J}} (r_j + p_j)$, then node $\sigma 45$ is removed according to R_2.

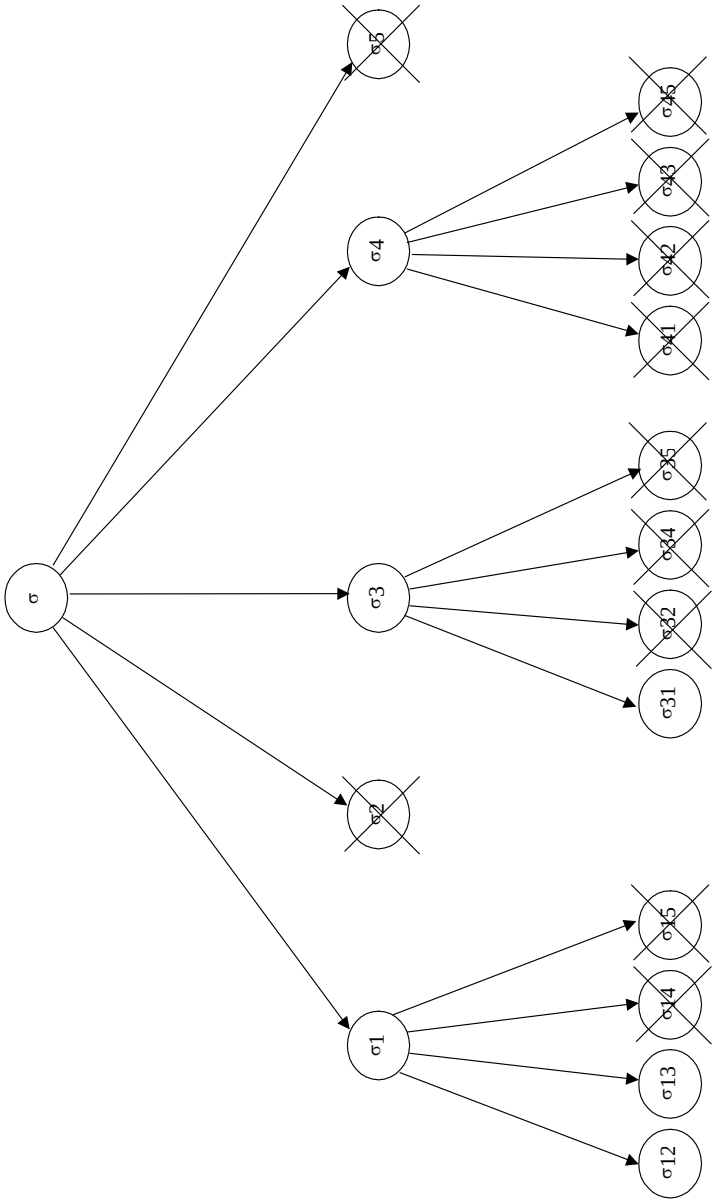


Figure 6.4. Application of the node selection rules

6.5 The extended semi-preemptive lower bound

In this section, we describe how the semi-preemptive lower bound (cf. page 49) can be extended in order to take into account in a more effective way the machine availability times. Firstly, the machine availabilities have to be considered in constructing the time intervals. That is, $e_1, e_2, ..., e_K$ are the different values of r_j $(j \in \bar{J})$, d_j $(j \in \bar{J})$, $d_j - p_j$ $(j \in S)$, $r_j + p_j$ $(j \in S)$ and u_i $(i = 1, ..., m)$ ranked in increasing order, where $S = \{j \in \bar{J}; d_j - r_j < 2p_j\}$. Secondly, note that a job j cannot be processed on a machine M_i such that $u_i + p_j > d_j$ in a nonpreemptive schedule. Thus, for each job j, we can determine the total number of machines α_j that do not satisfy $u_i + p_j > d_j$. Clearly, the semi-preemptive lower bound is improved if the above constraint is taken into account. For that purpose, it suffices to perform the following slight modification on the corresponding network.

Each interval node E_k is replaced by m_k interval nodes $E_k^1, E_k^2, ..., E_k^{m_k}$. Each interval node E_k^h $(h = 1, ..., m_k)$ represents the time interval $[e_k, e_{k+1}]$ but with h available machines. The interval node E_k^h is connected with E_k^{h+1} by an arc with capacity $h(e_{k+1} - e_k)$. The interval node $E_k^{m_k}$ is connected to the sink node with an arc with capacity $m_k(e_{k+1} - e_k)$. An arc (J_j, E_k) is replaced by an arc (J_j, E_k^h) with capacity $e_{k+1} - e_k$, where h is the largest integer in $[1, m_k]$ which is smaller than α_j.

The obtained lower bound will be referred to as the extended semi-preemptive lower bound and will be denoted by $SPLB^*$.

Example 6.4. Consider the feasibility problem defined on the 5 job - 3 machine instance which data are depicted in Table 6.3.

j	1	2	3	4	5
r_j	2	2	2	2	2
p_j	3	4	2	2	4
d_j	13	13	13	13	16

Table 6.3. Data of the 5 job - 3 machine instance of example 6.4

Assume that the machine availabilities are $\{2, 11, 12\}$. Then, we have

$$\alpha_1 = 1, \alpha_2 = 1, \alpha_3 = 2, \alpha_4 = 2, \alpha_5 = 3$$

The time intervals corresponding to this problem are $E_1 = [2, 11]$, $E_2 = [11, 12]$, $E_3 = [12, 13]$ and $E_4 = [13, 16]$. Their respective availabilities are

1, 2, 2 and 3. The flow networks corresponding to the (classical) semi-preemptive bound and the extended semi-preemptive bound are illustrated in Figures 6.5 and 6.6, respectively. Note that the time intervals E_4^1 and E_4^2 have not been depicted since they are not connected to any job node.

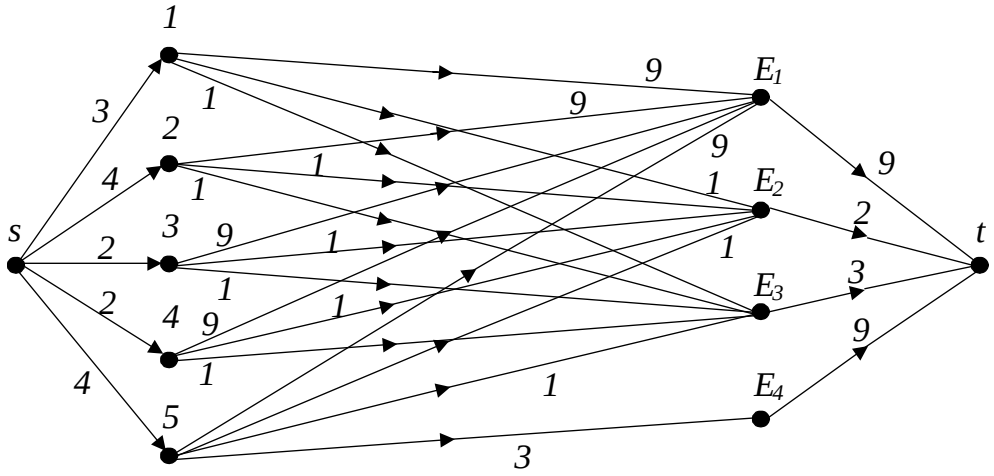


Figure 6.5. The flow network corresponding to $SPLB$

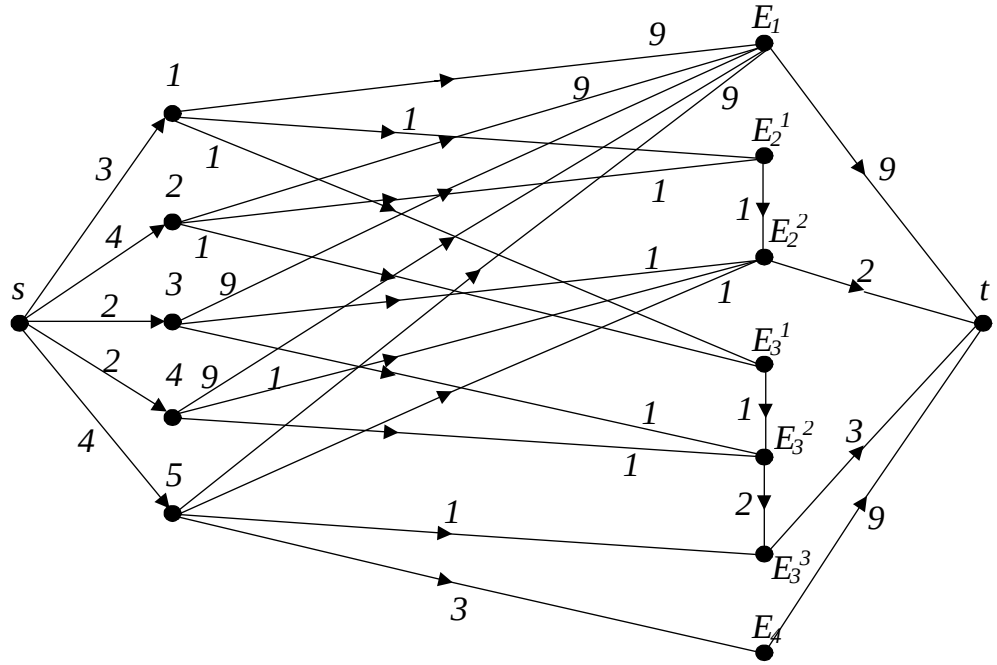


Figure 6.6 The flow network corresponding to $SPLB^*$

6.6 Upper bounds

In this section, two new families of approximate schedules are proposed. The first one is immediately derived from the optimal extended semi-preemptive schedule. The second one uses the FAP to check a potential infeasibility in the constructed schedule.

6.6.1 Semi-preemptive-based upper bounds

It is worth noting that the optimal extended semi-preemptive schedule described in the previous section provides a strong relaxation of the problem. Thus, it seems reasonable to consider that the job ordering in an optimal nonpreemptive schedule is "close" to that in an optimal extended semi-preemptive one. Therefore, an interesting feasible nonpreemptive schedule can be derived by considering the sequence obtained by sorting the jobs according to the nondecreasing order of their earliest starting times in the optimal extended semi-preemptive schedule (ties are settled according to the nonincreasing order of the tails). The makespan of the obtained schedule will be denoted by $SPUB(\bar{J})$.

Example 6.5. Consider the 7 job - 2 machine instance defined by Table 6.4.

j	1	2	3	4	5	6	7
r_j	0	4	3	4	0	1	3
p_j	4	3	3	7	3	8	3
q_j	2	7	8	1	6	7	1

Table 6.4. Data of the 7 job - 2 machine instance of example 6.5

Assume that we have $LB = 17$ and $UB = 19$. An optimal extended semi-preemptive schedule is depicted in Figure 6.7.

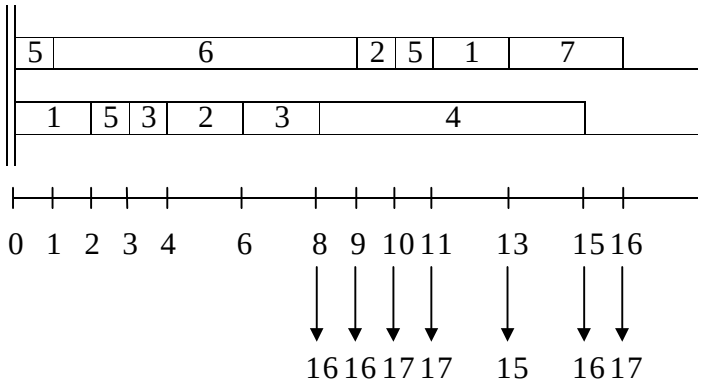


Figure 6.7. An optimal extended semi-preemptive schedule of example 6.5

Sorting the jobs according to their earliest starting times in this schedule yields the sequence $\sigma = (5, 1, 6, 3, 2, 4, 7)$. The nonpreemptive schedule corresponding to σ is depicted in figure 6.8. Its makespan equals 18.

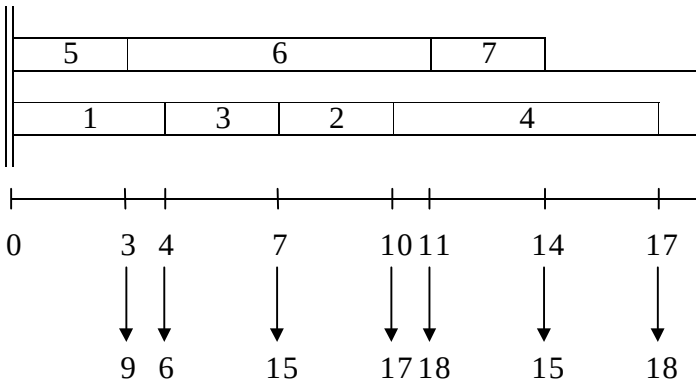


Figure 6.8. $SPUB(\bar{J}) = 18$

Let C_j and $f_j = C_j - q_j - p_j$ denote the completion time and the latest possible starting time of job $j \in \bar{J}$ in the optimal extended semi-preemptive schedule, respectively. A second approximate schedule can be derived by considering the sequence obtained by sorting the jobs according to the nondecreasing order of their f_j (ties are settled according to the nonincreasing order of the tails). The makespan of the obtained schedule will be denoted by $SPUB^{-1}(\bar{J})$.

Example 6.6. Consider the 7 job - 2 machine instance defined by Table 6.5.

j	1	2	3	4	5	6	7
r_j	5	4	6	3	6	4	2
p_j	4	7	6	6	5	1	7
q_j	6	2	7	1	6	3	6

Table 6.5. Data of the 7 job - 2 machine instance of example 6.6

Assume that we have $LB = 22$ and $UB = 25$. An optimal extended semi-preemptive schedule is depicted in Figure 6.9.

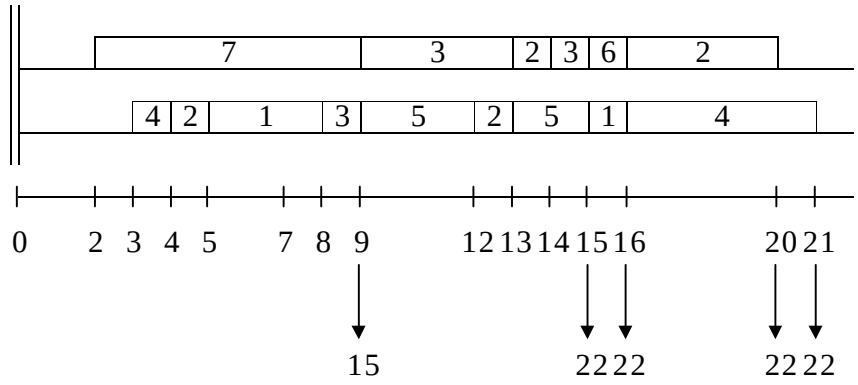


Figure 6.9. An optimal extended semi-preemptive schedule of example 6.6

The latest possible starting times of the jobs are

$$f_1 = 12, f_2 = 13, f_3 = 9, f_4 = 15, f_5 = 10, f_6 = 15, f_7 = 2$$

Sorting the jobs according to their latest possible starting times yields the sequence $\sigma = (7, 3, 5, 1, 2, 6, 4)$. The nonpreemptive schedule corresponding to σ is depicted in figure 6.10. Its makespan equals 24.

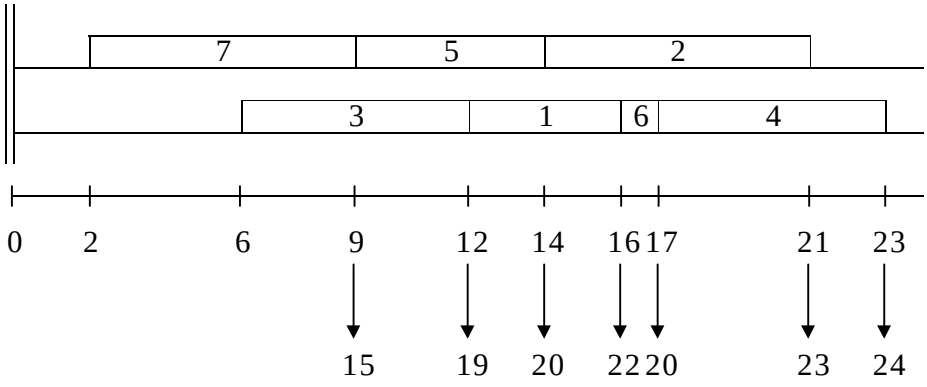


Figure 6.10. $SPUB^{-1}(\bar{J}) = 24$

6.6.2 Feasibility-based upper bounds

Although Jackson's schedule is a fairly good approximation schedule, its shortsightedness constitutes its major flaw. In this section we describe how we can use the FAP in order to anticipate at best the impact of the current decision. Assume that we are interested to construct a nonpreemptive schedule with makespan less than or equal to a trial value $C \in [LB, UB-1]$. First, we set $d_j = C - q_j$ for all $j \in \bar{J}$ and we adjust the heads and the tails using FAP. A job $j \in \bar{J}$ such that $d_j = r_j + p_j$ is referred to as a fixed job and is considered as already scheduled. Let L denote the list of the free (non-fixed) jobs in $\bar{J}$ sorted according to the nondecreasing order of their heads, where ties are settled according to the nonincreasing order of tails. At each iteration, we use FAP to check whether the first job $j_0 \in L$ can be scheduled at its release date (note that after applying FAP, all the release dates are larger than the smallest machine availability). In this case, we set $d_{j_0} = r_{j_0} + p_{j_0}$. The list L is then updated by the FAP.

Now, assume that the FAP yields an infeasibility. That is, scheduling j_0 at this position is not the right decision. Therefore, we have to skip job j_0 and move to the next job in the list. Note that there may be no possible job to be scheduled at the current iteration. In this case, finding a schedule with makespan less than or equal to the trial value C is probably impossible. So we have to move on to $C+1$ and so on. The algorithm stops when a feasible schedule is constructed. In the sequel, the makespan of this approximate schedule will be denoted by $FAP\_UB(\bar{J})$.

Example 6.7. Consider the 6 job - 3 machine instance defined by Table 6.6. The machines are assumed to be available from time zero onwards.

j	1	2	3	4	5	6
r_j	59	17	42	16	99	6
p_j	60	73	84	77	40	98
q_j	72	33	89	17	84	47

Table 6.6. Data of the 6 job - 3 machine instance of example 6.7

Assume that $LB = 228$ and $UB = 229$. Thus, we are interested in constructing a schedule with makespan equal to 228. Therefore, the deadlines are set to

$$d_1 = 156, d_2 = 195, d_3 = 139, d_4 = 211, d_5 = 144, d_6 = 181$$

Note that applying FAP to this instance yields $r_5 = 104$, which already fixes job 5 (i.e. $r_5 + p_5 = d_5$). Thus, the list of the free jobs is $L = \{6, 4, 2, 3, 1\}$.

At the first iteration, job 6 is scheduled at time $r_6 = 6$. That is, its deadline is set to $d_6 = 104$. The current data remain unchanged if FAP is applied. Thus, we have $L = \{4, 2, 3, 1\}$.

Assume that job 4 is scheduled at time $r_4 = 16$ at the second iteration. That is, its deadline is set to $d_4 = 93$. Applying FAP to the obtained instance yields an infeasibility. Indeed, during the time interval $[55, 93]$, the three machines are necessarily processing jobs 6, 4 and 3. Thus, the minimum starting time of job 1 is $r_1 = 93$. Which makes the three machines loaded during the time interval $[96, 126]$. That is, the minimum starting time of job 2 is $r_2 = 126 < d_2 - p_2 = 122$ which is impossible. Consequently, scheduling job 4 at the second iteration is not the right choice.

Now assume that job 2 (the next job in L) is scheduled instead at time $r_2 = 17$ at the second iteration. That is, its deadline is set to $d_2 = 90$. The modified data obtained after applying FAP are $r_1 = 90$, $r_4 = 126$, $d_3 = 134$ and $q_3 = 94$. Therefore, we have $L = \{3, 1, 4\}$ and so on. The obtained schedule is depicted in Figure 6.11. Its makespan is equal to 228.

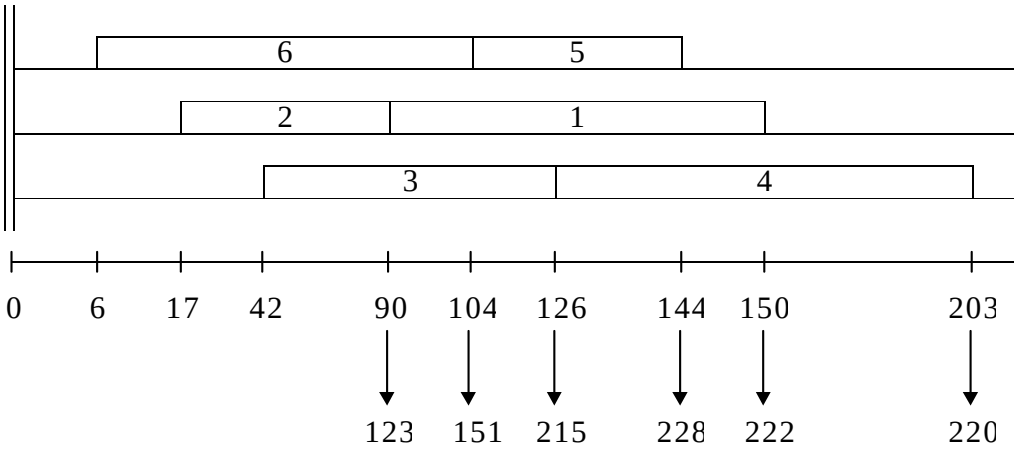


Figure 6.11. $FAP\_UB(\bar{J}) = 228$

A second FAP-based upper bound can be derived using the symmetry of the problem. That is, the schedule is constructed by starting from the end. The procedure is very similar to that of FAP_UB. The main difference is that the list L contains the free jobs in $\bar{J}$ sorted according to the nondecreasing order of their tails, where ties are settled according to the nonincreasing order of heads. Also, for a potential job j_0 to be scheduled, we set $r_{j_0} = d_{j_0} - p_{j_0}$. The obtained approximate makespan is denoted by $FAP\_UB^{-1}(\bar{J})$.

Example 6.8. Consider the 5 job - 2 machine instance defined by Table 6.7. The machine availabilities are equal to $\{111, 124\}$.

j	1	2	3	4	5
r_j	111	111	111	111	111
p_j	73	51	45	69	69
q_j	10	50	43	60	2

Table 6.7. Data of the 5 job - 2 machine instance of example 6.8

Assume that $LB = 277$ and $UB = 279$. We are interested in constructing a schedule with makespan equal to 278. Therefore, the deadlines are set to

$$d_1 = 268, d_2 = 228, d_3 = 234, d_4 = 218, d_5 = 276$$

The list of the free jobs is $L = \{5, 1, 3, 2, 4\}$. At the first iteration, job 5 is scheduled to finish at time $d_5 = 276$. That is, its head is set to $r_5 = 206$. The current data remain unchanged if FAP is applied. Thus, we have $L = \{1, 3, 2, 4\}$.

At the second iteration, job 1 is scheduled to finish at $d_1 = 268$. That is its head is set to $r_1 = 195$. Applying FAP yields $d_2 = d_3 = d_4 = 207$ and $q_2 = q_3 = q_4 = 70$. The list of the free jobs is $L = \{4, 2, 3\}$.

Assume that job 4 is scheduled to finish at time $d_4 = 207$ at the third iteration. That is, its head is set to $r_4 = 138$. Applying FAP to the obtained instance yields an infeasibility. Indeed, the two machines are loaded during the time interval $[195, 268]$. Thus, the maximum finishing time of job 3 is $d_3 = 195$. Which makes the two machines loaded during the time interval $[144, 162]$ by jobs 3 and 4. That is, the maximum finishing time of job 2 is $d_2 = 144 < r_2 + p_2 = 156$ which is impossible. Consequently, scheduling job 4 at the third iteration is not the right choice.

Now assume that job 2 (the next job in L) is scheduled instead to finish at time $d_2 = 207$ at the third iteration. That is, its head is set to $r_2 = 156$. The modified data obtained after applying FAP are $d_3 = 156$, $q_3 = 121$, $d_4 = 195$, $q_4 = 82$ and $r_4 = 124$. Therefore, job 3 is fixed and the only remaining job in the list is job 4. The obtained schedule is depicted in Figure 6.12. Its makespan is equal to 278.

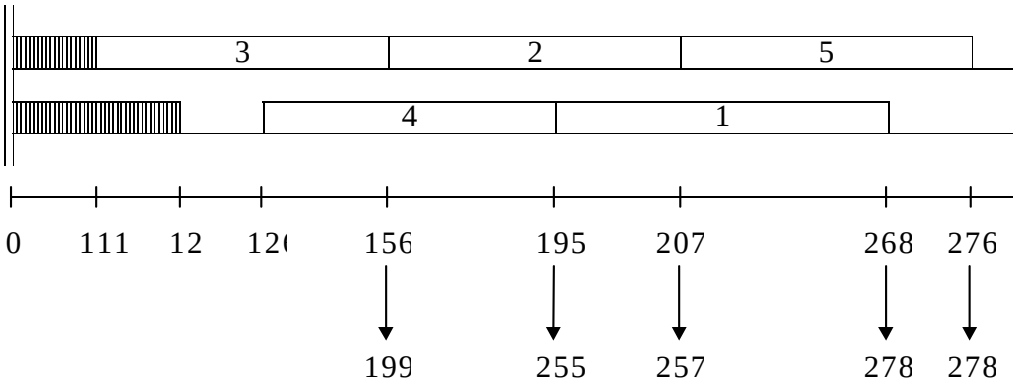


Figure 6.12 $FAP\_UB^{-1}(\bar{J}) = 278$

6.7 Synthesis of the branch-and-bound algorithm

Note that the problem defined at each node of the tree (except the root node) is a $P, NC_{inc}|r_j, q_j|C_{\max}$. Clearly, the preprocessing rules presented in Chapter 3 (page 28) do not apply for this problem. Therefore, the preprocessing algorithm is only used at the root node.

For a node N of the tree, the computed lower and upper bound are:

$$LB(N) = \max\{LB_0(J), SPLB^*(\overline{J})\}$$

$$UB(N) = \min\left\{SPUB(\bar{J}), SPUB^{-1}(\bar{J}), FAP\_UB(\bar{J}), FAP\_UB^{-1}(\bar{J})\right\}$$

Moreover, we implemented in the root node the additional lower bound $JMLB_3(\bar{J})$ and upper bound $\min\left\{JS(J), JS(\overline{J}), JS^{-1}(J), JS^{-1}(\overline{J})\right\}$.

In the following pseudo-code description of our sequence-based branch-and-bound algorithm, we adopted the following notation:

- N_p : the parent node of N
- $\Omega(N)$: the set of candidate descendant nodes of N
- N_0 : the current node to be branched.

It is worth noting that a Forward-Backward version of our algorithm has been implemented (see page 103). However, for the sake of clarity, it has not been mentioned in the following description.

Step 0. Initialization

0.1. *Make a root node R containing the data set of the problem.*

0.2. *Apply the preprocessing algorithm to node R.*

0.3. *Compute $UB(R)$. Set $UB = UB(R)$.*

0.4. *If $\left|\bar{J}\right| \leq m$, then go to Step 5. Else, compute $LB(R)$.*

0.5. *If $LB(R) = UB$, then go to Step 5. Else, compute $\Omega(R)$ using the node selection rules and set $N_0 = R$.*

Step 1. Node selection

If $\Omega(N_0) \neq \emptyset$, then select a node $N \in \Omega(N_0)$ and go to Step 2. Else, go to Step 4

Step 2. Branching

2.1 *Create the data of node N as described in Section 6.3.*

2.2. *Compute $LB(N)$. If $LB(N) \geq UB$ then go to step 3. Else set $q_j = \max\{q_j, LB(N) - d_j\}$ for all $j \in \bar{J}$.*

2.3. *Compute $UB(N)$. If $UB(N) < UB$ then set $UB = UB(N)$ and apply FAP to node N.*

2.4. *Compute $\Omega(N)$ using the node selection rules*

2.5. *Set $N_0 = N$ and go to step 1.*

Step 3. Pruning

Prune N from $\Omega(N_0)$ and go to step 1.

Step 4. Backtracking

If $N_0 = R$ then go to Step 5. Else, set $N = N_0$, $N_0 = N_p$ and go to step 3.

Step 5. Optimal makespan *Set $C^*_{\max} = UB$. Stop.*

6.8 Computational experiments

In this section we present the results of an analysis of the performance of the proposed sequence-based branch-and-bound algorithm ($B\&B2$) with our time-windows-based branch-and-bound algorithm ($B\&B1$). The two algorithms are coded in C and compiled with Visual C++ 5.0. All the computational experiments were carried out on a Pentium IV 2 GHz Personal Computer with 256 MB RAM.

6.8.1 Comparative study of the algorithms

The test problems are generated in a similar way as in Chapter 5. The number of jobs n is taken equal to 50, 100, 150, 200, 250 and 300 jobs. The number of machines m is taken equal to 2, 3, and 4 machines. The processing times are drawn from the discrete uniform distribution on [1,10]. The heads and tails are drawn from the discrete uniform distribution on $[1, K\frac{n}{m}]$, where K is a positive integer set equal to 1, 3, 5, and 7. We combined these problem characteristics to obtain 72 classes of randomly generated test problems. For each class, 10 instances are generated which results in 720 test problems. A CPU time limit of 300 sec. is fixed for each run.

Likewise in Chapter 5, for each algorithm we provide:

- $Time$: mean CPU time (in sec.)
- NN_{mean} : mean number of nodes
- $NN_{\max}$: maximal number of nodes
- US : percentage of instances for which optimality was not proved after reaching the time limit
- Gap_{mean} : average gap of unsolved instances, where the gap is defined as the ratio
$$100(\frac{UB - LB}{LB})$$
- $Gap_{\max}$: maximal gap of unsolved instances

Table 6.8 shows the remarkable performance of the sequence-based algorithm ($B\&B2$). Indeed,this algorithm consistently outperforms the time-windows-based one $B\&B1$ (and therefore Carlier's algorithm). We observe

that $B\&B2$ solved all of the 720 instances within the time limit of 300 seconds. It is worth noting that the *maximal* computing time of $B\&B2$ is only 7.12 seconds. The mean CPU time of $B\&B2$ is 0.16 sec., and this algorithm is 125.25 times faster than $B\&B1$ and 459.93 times faster than Carlier's algorithm. Moreover, the *maximal* number of nodes explored by $B\&B2$ is 26.03 times less than the *average* number of nodes explored by $B\&B1$.

	$Time$	NN_{mean}	$NN_{\max}$	US	Gap_{mean}	$Gap_{\max}$
$B\&B1$	20.04	963.45	48632	6.11	0.68	1.51
$B\&B2$	0.16	1.17	37	0.00	-	-

Table 6.8. Performance of the algorithms

6.8.2 Solution of large-sized instances

Motivated by the good performance of our sequence-based algorithm, we run large test problems generated as follows. The number of jobs n is taken equal to 500, 700, 1000, 1200 and 1500 jobs. The number of machines m is taken equal to 10, 20, 30, 40 and 50 machines. The processing times are drawn from the discrete uniform distribution on [1,10]. The heads and tails are drawn from the discrete uniform distribution on $[1, \frac{n}{m}]$. For each combination of n and m, 10 instances are generated which results in 250 test problems. The CPU time limit has been kept equal to 300 sec. for each run.

Note that this new set of instances belong to the most difficult class observed in Chapter 5 ($K = 1$ and large number of machines). We recall that 40% of the instances with only 4 machines remained unsolved within the time limit of 300 seconds by $B\&B1$. Compared to these small-sized instances, the new set of test problems is considered as very large-sized ones.

Table 6.9 shows that $B\&B2$ makes it feasible to solve 94.40% of the large-sized instances within less than one minute, on average. Moreover, for the unsolved instances, the algorithm provides an approximate solution which is, on average, no more than 0.52% from the optimum.

$Time$	NN_{mean}	$NN_{\max}$	US	Gap_{mean}	$Gap_{\max}$
54.71	1.37	21	5.60	0.52	0.75

Table 6.9. Performance of $B\&B2$ on large-sized instances

Tables 6.10 and 6.11 show the sensitivity of $B\&B2$ to the variation of n and m. We observe that the mean number of explored nodes is nearly steady for all the problem sizes. However, the CPU time required by $B\&B2$ seems

to be more sensitive to the variation of n than to the variation of m. Not surprisingly, the percentage of unsolved instances within the time limit tends to increase as n or m increases. Indeed, the more difficult instances are those with 1500 jobs and 50 machines. It is worth noting that for the unsolved instances of this large-sized class, $B\&B2$ provided a maximal gap of 0.60% from the optimum.

	$Time$	NN_{mean}	$NN_{\max}$	US	Gap_{mean}	$Gap_{\max}$
$n = 500$	5.18	1.16	5	0.00	-	-
$n = 700$	27.21	1.78	21	4.00	0.75	0.75
$n = 1000$	41.41	1.28	9	4.00	0.63	0.70
$n = 1200$	81.76	1.36	5	6.00	0.34	0.59
$n = 1500$	117.99	1.28	3	14.00	0.50	0.60

Table 6.10. Sensitivity of $B\&B2$ to the variation of n

	$Time$	NN_{mean}	$NN_{\max}$	US	Gap_{mean}	$Gap_{\max}$
$m = 10$	48.28	1.20	5	2.00	0.15	0.15
$m = 20$	49.51	1.28	5	4.00	0.26	0.29
$m = 30$	63.57	1.84	21	8.00	0.60	0.75
$m = 40$	45.11	1.14	3	4.00	0.65	0.70
$m = 50$	67.08	1.40	4	10.00	0.59	0.60

Table 6.11. Sensitivity of $B\&B2$ to the variation of m

Conclusion and directions for future research

In this thesis, we presented two branch-and-bound algorithms for the exact resolution of the identical parallel machine scheduling problem with heads and tails, denoted by $P|r_j, q_j|C_{\max}$. The importance of this strongly $\mathcal{NP}$-hard problem is twofold. Firstly, it generalizes several well-known parallel machine scheduling problems. Secondly, it plays a crucial role in solving more complex scheduling problems. Moreover, this problem arises in several practical applications. The only algorithm proposed to solve exactly the $P|r_j, q_j|C_{\max}$ is the one presented by Carlier (1987).

A new preprocessing algorithm devised to simplify the problem by reducing the number of unscheduled jobs is proposed. This algorithm plays a crucial role in accelerating the convergence of the proposed branch-and-bound algorithms. New lower bounding strategies are investigated for the $P|r_j, q_j|C_{\max}$. In particular, a new bounding scheme based on relaxing the problem as a parallel machine scheduling problem with availability constraints is proposed. Also, a new bin-packing based lower bound, as well as several new lifting procedures are derived. Moreover, we introduce the new concept of semi-preemptive scheduling and we show how it can be used to derive a maximum-flow-based lower bound for the $P|r_j, q_j|C_{\max}$ which dominates the well-known preemptive lower bound. Extensive numerical experiments show that the proposed lower bounds consistently outperform the best existing ones.

The first proposed branch-and-bound algorithm is based on time windows. This algorithm embodies the best derived lower bound as well as the proposed preprocessing algorithm. Also, new procedures that construct improved approximate schedules are incorporated in the algorithm. Moreover, an effective algorithm that aims at adjusting the parameters of the problem and identifying the inconsistencies as well as strengthening the lower bounds is embedded in the branch-and-bound algorithm. Furthermore, an original pseudo-parallel implementation based on the symmetry of the $P|r_j, q_j|C_{\max}$ is proved to be very useful for the efficiency of the algorithm. Extensive computational experiments show that although the two algorithms are time-windows-based, our algorithm consistently outperforms Carlier's one.

The second proposed branch-and-bound algorithm is based on a new original representation of the $P|r_j, q_j|C_{\max}$ as a sequencing problem. Indeed, we

show that specifying an order between the different jobs suffices to build a feasible schedule. This problem formulation enables us to derive several rules that consistently reduce the size of the search tree. Also, we show how a strong network-flow-based relaxation scheme can be derived through a generalization of the semi-preemptive lower bound. Moreover, new effective heuristics are shown to be of a capital importance for the efficiency of the branch-and-bound algorithm. Our experimental results show that this new algorithm makes it possible to solve up to 1500 jobs-50 machine instances in a small CPU time, whereas previous algorithms could not even solve instances with 50 jobs and 2 machines. This result constitutes a significant step toward efficient solutions of $\mathcal{NP}$-hard parallel machine scheduling problems.

Future research effort needs to be focused on solving more complex scheduling problems. In particular, an issue worthy of future investigation, is to embed the findings of this thesis in investigating exact or approximate procedures for the multiprocessor job shop and the resource constrained project scheduling. Also, the present work could be extended to the case of uniform and unrelated machines.

Bibliography

[1] **J.O. Achugbue, F.Y. Chin** (1981), "*Bounds on schedules for independent tasks with similar execution times",* Journal of the Association of Computer Machinery 28, 81-99.

[2] **J. Adams, E. Balas, D. Zawack (**1988), "*The shifting bottleneck procedure for job shop scheduling",* Management Science 34, 391-401.

[3] **A.S. Alfa, K.L. Dolhun, C.-L. Li** (1994), "*A Scheduling Problem in a Semi-Open Job Shop - The Case of a Cordset Manufacturing Company",* Mathematics and Computer Modelling 20, 1-8.

[4] **K.R. Baker** (1974), "*Introduction to Sequencing and Scheduling",* Wiley.

[5] **E. Balas** (1967), "*Discrete programming by the filter method",* Operations Research 15, 915.

[6] **J. Barnes, M. Laguna, F. Glover** (1995), "*An overview of tabu search approaches to production scheduling problems*", in: D. Brown and W. Scherer (Eds.), Intelligent scheduling systems, Kluwer Academic Publishers, Boston, 101-128.

[7] **C. Basnet, L.R. Foulds, J.M. Wilson** (1999), "*An exact algorithm for a milk tanker scheduling and sequencing problem*", Annals of Operations Research 86, 559-568.

[8] **A.A. Bertossi, A. Fusiello** (1997), "*Rate-monotonic scheduling for hard-real-time systems",* European Journal of Operational Research 96, 429-443.

[9] **J. Blazewicz** (1987), "*Selected topics in scheduling theory",* Annals of Discrete Mathematics 31, 1-60.

[10] **J. Blazewicz, G. Finke, R. Haupt, G. Schmidt** (1988), "*New trends in machine scheduling",* European Journal of Operational Research 37, 303-317.

[11] **J. Blazewicz, M. Dror, J. Weglarz** (1991), "*Mathematical programming formulations for machine scheduling : A survey",* European Journal of Operational Research 51, 283-300.

[12] **J. Blazewicz, M. Drozdowski, J. Weglarz** (1994a), "*Scheduling multiprocessor task: a survey"*, International Journal on Microcomputer Applications 13, 89-97.

[13] **J. Blazewicz, K.H. Ecker, G. Schmidt, J. Weglarz** (1994b), "*Scheduling in Computer and Manufacturing Systems",* Springer-Verlag.

[14] **J.D. Blocher, S. Chand** (1991), "*Scheduling of parallel processors: A posteriori bound on LPT sequencing and a two-step algorithm",* Naval Research Logistics 38, 273-287.

[15] **P. Bratley, M. Florian, P. Robillard** (1975), "*Scheduling with earliest start and due date constraints on multiple machines",* Naval Research Logistics Quarterly 22, 165-173.

[16] **P. Brucker, J. Hurink, F. Werner** (1996), "*Improving local search heuristics for some scheduling problems. Part I",* Discrete Applied Mathematics 65, 97-122.

[17] **P. Brucker, J. Hurink, F. Werner** (1997), "*Improving local search heuristics for some scheduling problems. Part II",* Discrete Applied Mathematics 72, 47-69.

[18] **P. Brucker** (1998), "*Scheduling Algorithms",* Springer.

[19] **P. Brucker, J. Hurink** (2000), "*Solving a chemical batch scheduling problem by local search",* Annals of Operations Research 96, 17-38.

[20] **R.E. Burkard, Y. He** (1998), "*A note on MULTIFIT scheduling for uniform machines",* Computing 61, 277-283.

[21] **G. Buxey** (1989), "*Production scheduling : Practice and theory",* European Journal of Operational Research 39, 17-31.

[22] **J. Carlier** (1982), "*The one machine sequencing problem",* European Journal of Operational Research 11, 42-47.

[23] **J. Carlier** (1987), "*Scheduling jobs with release dates and tails on identical machines to minimize the makespan",* European Journal of Operational Research 29, 298-306.

[24] **J. Carlier, B. Latapie** (1991), "*Une méthode arborescente pour résoudre les problèmes cumulatifs"*, RAIRO 25, 311-340.

[25] **J. Carlier, E. Pinson** (1998), "*Jackson's Pseudo Preemptive Schedule for the $Pm/r_j, q_j/C_{\max}$ scheduling problem",* Annals of Operations Research ; 83, 41-58.

[26] **B. Chen** (1991), "*Tighter bound for MULTIFIT scheduling on uniform processors",* Discrete Applied Mathematics 31, 227-260.

[27] **T.C.E. Cheng, C.C.S. Sin** (1990), "*A state-of-the-art review of parallel-machine scheduling research",* European Journal of Operational Research 47, 271-292.

[28] **Y. Cho, S. Sahni** (1980), "*Bounds for list schedules on uniform processor"*, SIAM Journal on Computing 9, 91-103.

[29] **P. Chrétienne, C. Picouleau** (1995), "*Scheduling with communication delays: A survey*", in: P. Chrétienne, E.G., Jr., Coffman, J.K. Lenstra and Z. Liu (eds.), Scheduling Theory and its Applications, Wiley, New York, 65-90.

[30] **E.G. Coffman Jr., M.R. Garey, D.S. Johnson** (1978), "*An application of bin-packing to multiprocessor scheduling",* SIAM Journal on Computing 7, 1-17.

[31] **R.W. Conway, W.L. Maxwell, L.W. Miller** (1967), "*Theory of scheduling",* Addison-Wesley.

[32] **E. Davis, J.M. Jaffe** (1981), "*Algorithms for scheduling tasks on unrelated processors",* Journal of the Association for Computing Machinery 28, 721-736.

[33] **M. Davis, N. Aquilano, R. Chase** (1999), "*Fundamentals of operations management",* 3rd edition, Irwin-McGraw Hill.

[34] **M. Dell'Amico, S. Martello** (1995), "*Optimal scheduling of tasks on identical parallel processors",* ORSA Journal on Computing 7, 191-200.

[35] **F. Della Croce, R. Tadei, P.S. Asioli** (1999), "*Scheduling a round robin tennis tournament under courts and players availability constraints*", Annals of Operations Research 92, 349-361.

[36] **M.I. Dessouky, B.J. Lageweg, J.K. Lenstra, S.L. van de Velde** (1990), "*Scheduling identical jobs on uniform parallel machines*", Statistica Neerlandica 44, 115-123.

[37] **M.M. Dessouky (**1998**),** "*Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize the maximum lateness",* Computers and Industrial Engineering 34, 793-806.

[38] **G. Dobson** (1984), "*Scheduling independent tasks on uniform processors*", SIAM Journal on Computing 13, 705-716.

[39] **B. Dodin, K.H. Chan** (1991), "*Application of production scheduling methods to external and internal audit scheduling",* European Journal of Operational Research 52, 267-279.

[40] **J. Dorn, M. Girsch, G. Skele, W. Slany** (1996), "*Comparison of iterative improvement techniques for schedule optimization*", European Journal of Operational Research 94, 349-361.

[41] **M. Drozdowski** (1996), "*Scheduling multiprocessor tasks - An overview*", European Journal of Operational Research 94, 215-230.

[42] **A. Federgruen, H. Groenevelt** (1986), "*Preemptive scheduling of uniform machines by ordinary network flow techniques*", Management Science 32, 341-349.

[43] **J.A. Ferland, I. Berrada, I. Nabli, B. Ahiod, P. Michelon, V. Gascon, E. Gagné** (2001), "*Generalized assignment type goal programming problem: application to nurse scheduling*", Journal of Heuristics 7, 391-413.

[44] **J. Frenk, A.H.G. Rinnooy Kan** (1987), "*The asymptotic optimality of the LPT rule*", Mathematics of Operations Research 14, 241-254.

[45] **D.K. Friesen, M.A. Langston** (1983), "*Bounds for MultiFit scheduling on uniform processors*", SIAM Journal of Computing 12, 60-70.

[46] **D.K. Friesen** (1984), "*Tighter bounds for the MultiFit scheduling algorithm*", SIAM Journal on Computing 13, 170-181.

[47] **D.K. Friesen, M.A. Langston** (1986), "*Evaluation of a MULTIFIT-based scheduling algorithm*", Journal of Algorithms 7, 35-59.

[48] **D.K. Friesen** (1987), "*Tighter bounds for LPT scheduling on uniform processor*", SIAM Journal on Computing 16, 554-660.

[49] **M.R. Garey, D.S. Johnson** (1978), "*Strong NP-completeness results : Motivation, examples and implications*", Journal of the Association of Computer Machinery 25, 499-508.

[50] **M.R. Garey, D.S. Johnson** (1979), "*Computers and Intractability : A Guide to the Theory of NP-Completeness*", Freeman.

[51] **A. Gharbi, M. Haouari** (2002), "*Minimizing Makespan on Parallel Machines Subject to Release Dates and Delivery Times*", Journal of Scheduling 5, 329-355.

[52] **C.A. Glass, C.N. Potts, P. Shade** (1994), "*Unrelated Parallel Machine Scheduling Using Local Search*", Mathematical and Computational Modelling 20, 41-52.

[53] **F. Glover** (1989), "*Tabu Search. Part I*", ORSA Journal on Computing 1, 190-205.

[54] **F. Glover** (1990), "*Tabu Search. Part II*", ORSA Journal on Computing 2, 4-32.

[55] **F. Glover** (1996), "*Tabu Search and adaptive memory programming - Advances, applications and challenges*", Interfaces in Computer Science and Operations Research, Barr, Helgason and Kennington (eds.), Kluwer Academic Publishers, 1-75.

[56] **E. Goldratt, J. Cox** (1992), "*The Goal: a process of ongoing improvement",* 2nd rev, North River Press.

[57] **T. Gonzalez, O.H. Ibarra, S. Sahni** (1977), "*Bounds for LPT schedules on uniform processors*", SIAM Journal on Computing 6, 155-166.

[58] **T. Gonzalez, S. Sahni** (1978), "*Preemptive scheduling of uniform processor systems*", Journal of the Association of Computer Machinery 25, 92-101.

[59] **T. Gonzalez, D.B. Johnson** (1980), "*A new algorithm for preemptive scheduling of trees*", Journal of the Association for Computing Machinery 27, 287-312.

[60] **T. Gonzalez, E.L. Lawler, S. Sahni** (1990), "*Optimal preemptive scheduling of two unrelated processors"*, ORSA Journal on Computing 2, 219-224.

[61] **R.L. Graham** (1966), "*Bounds for certain multiprocessing anomalies*", Bell System Technical Journal 45, 1563-1581.

[62] **R.L. Graham** (1969), "*Bounds on multiprocessing timing anomalies"*, SIAM Journal on Applied Mathematics 17, 263-269.

[63] **R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan** (1979), "*Optimization and approximation in deterministic sequencing and scheduling: a survey*", Annals of Discrete Mathematics 5, 287-326.

[64] **D. Gusfield** (1984), "*Bounds for naive multiple machine scheduling with release times and deadlines",* Journal of Algorithms 5, 1-6.

[65] **C. Hanen, A. Munier** (1995), "*An approximation algorithm for scheduling dependent tasks on m processors with small communication delays"*, IEEE Symposium Emerging Technologies and Factory Automation, Paris ,167-189.

[66] **M. Haouari, A. Gharbi** (2003), "*An Improved Max-Flow Based Lower Bound for Minimizing Maximum Lateness on Identical Parallel machines*", Operations Research Letters 31, 49-52.

[67] **E. Hart, P. Ross, J.A.D. Nelson** (1999), "*Scheduling chicken catching - an investigation into the success of a genetic algorithm on a real-world scheduling problem*", Annals of Operations Research 92, 363-380.

[68] **R. Haupt** (1989), "*A survey of priority rule-based scheduling*", Operational Research Spektrum 11, 3-16.

[69] **J.C. Ho, J.S. Wong** (1995), "*Makespan minimization for m parallel identical processors*", Naval Research Logistics 42, 935-948.

[70] **D.S. Hochbaum, D.B. Shmoys** (1987), "*Using dual approximation algorithms for scheduling problems: theoretical and practical results*", Journal of the Association for Computing Machinery 34, 144-162.

[71] **K. Hoffman, M. Padberg** (1991), "*Improving LP-representations of zero-one linear programs for branch-and-cut".* ORSA Journal on Computing 3, 121-135.

[72] **J.A. Hoogeveen, J.K. Lenstra, B. Veltman** (1994), "*Three, four, five, six, or the complexity of scheduling with communication delays*", Operations Research Letters 16, 129-137.

[73] **H. Hoogeveen, C. Hurkens, J.K. Lenstra, A. Vandevelde** (1995), "*Lower bounds for the multiprocessor flow shop*", Second Workshop on Models and Algorithms for Planning and Scheduling, Wernigerode.

[74] **W.A. Horn** (1974), "*Some simple scheduling algorithms",* Naval Research Logistics Quarterly 21, 177-185.

[75] **E. Horowitz, S. Sahni** (1976), "*Exact and approximate algorithms for scheduling nonidentical processors*", Journal of the Association for Computing Machinery 23, 317-327.

[76] **R. Hübscher, F. Glover** (1994), "*Applying Tabu Search with influential diversification to multiprocessor scheduling*", Computers Operations Research 8/21, 877-884.

[77] **O.H. Ibarra, C.E. Kim** (1977), "*Heuristic algorithms for scheduling independent tasks on nonidentical processors*", Journal of the Association for Computing Machinery 24, 280-289.

[78] **J.R. Jackson** (1955), "*Scheduling a production line to minimize maximum tardiness"*, Management Science Research Project, University of California, Los Angeles.

[79] **J.R. Jackson** (1956), "*An extension of Johnson's results on job lot scheduling*", Naval Research Logistics Quarterly 3, 201-203.

[80] **A.I.Z. Jarrah, J.F. Bard, A.H. de Silva** (1992), "*A heuristic for machine scheduling at general mail facilities",* European Journal of Operational Research 63, 192-206.

[81] **S.M. Johnson** (1954), "*Optimal two- and three-stage production schedules with setup times included",* Naval Research Logistics Quarterly 1, 61-67.

[82] **J. Jozefowska, M. Milka, R. Rozycki, G. Waligora, J. Weglarz** (1998), "*Local search metaheuristics for discrete-continuous problems*", European Journal of Operational Research 107, 354-370.

[83] **R.M. Karp** (1972), "*Reducibility among combinatorial problems",* in Complexity of Computer Computations, R.E. Miller and J.W. Tatcher (eds.), Plenum Press, pp. 85-103.

[84] **H. Kellerer** (1998), "*Algorithms for multiprocessor scheduling with machine release times",* IIE Transactions 30, 991-999.

[85] **J. Labetoulle, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan** (1984), "*Preemptive scheduling of uniform machines subject to release dates"*, Progress in Combinatorial Optimization, Academic Press, Florida, 245-261.

[86] **G. Lancia** (2000), "*Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan",* European Journal of Operational Research 120, 277-288.

[87] **E.L. Lawler, J. Labetoulle** (1978), "*On preemptive scheduling of unrelated parallel processors by linear programming",* Journal of the Association for Computing Machinery 25, 612-619.

[88] **E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D. Shmoys** (1993), "*Sequencing and Scheduling : Algorithms and Complexity",* Handbooks in Operations Research and Management Science 4, S.S. Graves, A.H.G. Rinnooy Kan, P. Zipkin (eds.), 445-522.

[89] **C-Y. Lee, J.D. Massey** (1988b), "*Multiprocessor scheduling: combining LPT and MULTIFIT",* Discrete Applied Mathematics 20, 233-242.

[90] **C-Y. Lee** (1991), "*Parallel machine scheduling with non-simultaneous machine available time",* Discrete Applied Mathematics 30, 53-61.

[91] **J.K. Lenstra** (1977), "*Sequencing by Enumerative Methods",* Mathematical Center Tracts 69, Mathematisch Centrum, Amsterdam.

[92] **J.K. Lenstra, D.B. Shmoys, E. Tardos** (1990), "*Approximation algorithms for scheduling unrelated parallel machines*", Mathematical Programming 46, 259-271.

[93] **J.W.S. Liu, C.L. Liu** (1974), "*Performance analysis of heterogeneous multiprocessor computing systems*", in: E. Gelenbe, R. Mahl (Eds.), Computer Architecture and Networks, North Holland, Amsterdam, 331-343.

[94] **W. Ludwing, P. Tiwari** (1994), "S*cheduling malleable and nonmalleable parallel tasks"*, Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 167-176.

[95] **S. Martello, P. Toth** (1990), "*Knapsack problems: Algorithms and computer implementations"*, John Wiley & Sons.

[96] **S. Martello, F. Soumis, P. Toth** (1997), "*Exact and approximation algorithms for makespan minimization on unrelated parallel machines"*, Discrete Applied Mathematics 75, 169-188.

[97] **R. McNaughton** (1959), "*Scheduling with deadlines and loss function*", Management Science 6, 1-12.

[98] **M. Minoux** (1986), "*Mathematical Programming : Theory and Algorithms*", Wiley.

[99] **R.H. Möhring, M.W. Schäffter, A.S. Schulz** (1996), "*Scheduling jobs with communication delays: using infeasible solutions for approximation*", in: J. Diaz and M. Serna, (eds.), Algorithms-ESA '96, Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science 1136, Springer, Berlin ,76-90.

[100] **E. Mokotoff** (2001), "*Parallel Machine Scheduling Problems: A Survey*", Asia-Pacific Journal of Operations Research 18, 193-243.

[101] **E. Mokotoff, J. L. Jimeno, A. I. Gutiérrez** (2001), "*List Scheduling Algorithms to Minimize the Makespan on a identical Parallel Machines",* TOP 9/2, 243-269.

[102] **E. Mokotoff** (2002), "*An exact algorithm for the Identical Parallel Machine Scheduling Problem"*, submitted in European Journal of Operational Research.

[103] **E. Mokotoff, P. Chrétienne** (2002), "*A Cutting Plane Algorithm for the Unrelated Parallel Machine Scheduling Problem*", European Journal of Operational Research 141, 517-527.

[104] **E. Mokotoff, J. L. Jimeno** (2002), "*Heuristics Based on Partial Enumeration for the Unrelated Parallel Processor Scheduling Problem*", Annals of Operations Research (to appear).

[105] **J.F. Morrison** (1988), "*A note on LPT scheduling*", Operations Research Letters 7, 77-79.

[106] **T.E. Morton, D.W. Pentico** (1993), "*Heuristic Scheduling Systems*", Wiley.

[107] **A. Munier, J.C. König** (1997), "*A Heuristic for a Scheduling Problem with Communication Delays"*, Operations Research 45, 145-147.

[108] **I. Nabeshima, A.O. Esogbue, R. Bellman** (1982), "*Mathematical Aspects of Scheduling and Applications",* Pergamon Press.

[109] **G. Nemhauser, L. Wolsey** (1988), "*Integer and Combinatorial Optimization*", Wiley.

[110] **S.S. Panwalkar, W. Iskander** (1977), "*A survey of scheduling rules*", Operations Research 25, 45-61.

[111] **C.H. Papadimitriou** (1994), "*Computational Complexity"*, Addison-Wesley.

[112] **R.G. Parker** (1995), "*Deterministic Scheduling Theory*", Chapman & Hall.

[113] **P.C. Pendharkar, J.A. Rodger** (2000), "*Nonlinear programming and genetic search application for production scheduling in coal mines*", Annals of Operations Research 95, 251-267.

[114] **M. Perregaard (**1995), "*branch-and-bound Method for the Multiprocessor Jobshop and Flowshop Scheduling Problem"*, Master Thesis, Datalogisk institut KØbenhavns Universitet.

[115] **N. Piersma, W. van Dijk** (1996), "*A Local Search Heuristic for Unrelated Parallel Machine Scheduling with Efficient Neighborhood Search"*, Mathematical and Computational Modelling 24, 11-19.

[116] **M. Pinedo** (1995)**,** "*Scheduling : Theory, Algorithms, and Systems",* Prentice Hall.

[117] **M. Pinedo, X. Chao** (1998), "*Operations Scheduling With Applications in Manufacturing and Services*", Irwin.

[118] **C.N. Potts** (1985), "*Analysis of a linear programming heuristic for scheduling unrelated parallel machines",* Discrete Applied Mathematics 10, 155-164.

[119] **J. Riera, D. Alcaide, J. Sicilia** (1996), "*Approximate algorithms for the $P//C_{\max}$ problem*", Top 2/4, 345-359.

[120] **Y. Rochat** (1998), "*A genetic approach for solving a scheduling problem in a robotized analytical system",* Journal of Heuristics 4, 245-261.

[121] **E.S. Rosenbloom, N.F. Goertzen** (1987), "*Cyclic nurse scheduling*", European Journal of Operational Research 31, 19-23.

[122] **M.H. Rothkopf** (1966), "*Scheduling independent tasks on parallel processors*", Management Science 12, 437-447.

[123] **S. Sahni** (1976), "*Algorithms for scheduling independents tasks*", Journal of the Association for Computing Machinery 23, 116-127.

[124] **M.W.P. Savelsbergh** (1994), "*Preprocessing and probing techniques for mixed integer programming problems",* ORSA Journal on Computing 6, 445-454.

[125] **G. Schmidt** (1988), "*Scheduling independent tasks with deadlines on semi-identical processors",* Journal of Operational Research Society 39, 271-277.

[126] **G. Schmidt** (2000), "*Scheduling with limited machine availability",* European Journal of Operational Research 121, 1-15.

[127] **U. Schwiegelshohn, W. Ludwing, L. Wolf, J. Turek, P.S. Yu** (1997), "*Smart SMART bounds for weighted response time scheduling*", SIAM Journal on Computing.

[128] **M.G. Scutella, A. Frangioni, E. Necciari,** (2000), "*Multi-exchange algorithms for the minimum makespan machine scheduling problem*", 17 th European Conference on Operational Research, Budapest.

[129] **B. Simons, M. Warmuth** (1989), "*A fast algorithm for multiprocessor scheduling of unit-length jobs",* SIAM Journal on Computing 18, 690-710.

[130] **D. Sipper, R.Jr. Buflin (**1998), "*Production: planning, control, and integration",* McGraw-Hill.

[131] **W.E. Smith** (1956), "*Various opimizers for single stage production",* Naval Research Logistics Quarterly 3, 59-66.

[132] **F. Sourd** (2001), "*Scheduling tasks on unrelated machines: large neighborhood improvement procedures*", Journal of Heuristics 7, 519-531.

[133] **B. Srivastava** (1998), "*An Effective Heuristic for Minimising Makespan on Unrelated Parallel Machines*", Journal of the Operational Research Society 49, 886-894.

[134] **V.S. Tanaev, V.S. Gordon, Y.M. Shafransky** (1994a), "*Scheduling Theory: Single-Stage Systems*", Kluwer.

[135] **V.S. Tanaev, Y.N. Sotskov, V.A. Strusevich** (1994b), "*Scheduling Theory: Multi-Stage Systems*", Kluwer.

[136] **A. Thesen** (1998), "*Design and evaluation of tabu search algorithms for multiprocessor scheduling"*, Journal of Heuristics 4, 141-160.

[137] **J. Turek, W. Ludwing, L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, P.S. Yu** (1994), "*Scheduling parallelizable tasks to minimize average response time"*, Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures, 200-209.

[138] **J.D. Ullman** (1975), "*NP-complete scheduling problems",* Journal of Computer and System Sciences 10, 384-393.

[139] **S.L. van de Velde** (1993), "*Duality-based algorithms for scheduling unrelated parallel machines"*, ORSA Journal on Computing 5, 192-205.

[140] **A. Vandevelde** (1994), "*Minimizing the makespan in a multiprocessor flow shop*", Master's Thesis*,* Eindhoven University of Technology, The Netherlands.

[141] **B. Veltman, J.K. Lagegeweg, J.K. Lenstra** (1990), "*Multiprocessor scheduling with communication delays"*, Parallel Computing 16, 173-182.

[142] **H.M. Wagner** (1959), "*An integer linear programming model for machine scheduling",* Naval Research Logistics Quarterly 6, 131.

[143] **S.T. Webster** (1996), "*A general lower bound for the makespan problem",* European Journal of Operational Research 89, 516-524.

[144] **D. de Werra** (1990), "*Almost nonpreemptive schedules",* Annals of Operations Research 26, 243-256.

[145] **V.C.S. Wiers** (1997), "*A Review of the Applicability of OR and AI Scheduling Techniques in Practice",* Omega 25, 145-153.

[146] **J. Xu, S.Y. Chiu** (2001), "*Effective heuristic procedures for a field technician scheduling problem*", Journal of Heuristics 7, 495-509.

[147] **M. Yue** (1990), "*On the exact upper bound for the MultiFit processor scheduling algorithm*", Annals of Operations Research 24, 233-259.