New single machine scheduling problems with deadline for the characterization of optimal solutions

Thanh Thuy Tien Ta

UNIVERSITÉ DE TOURS

École Doctorale Mathématiques, Informatique, Physique Théorique, Ingénierie des Systèmes (MIPTIS)

Laboratoire d'Informatique Fondamentale et Appliquée de Tours (LIFAT, EA 6300)
Equipe Recherche Opérationnelle : Ordonnancement, Transport (ROOT, ERL CNRS 6305)

THÈSE présentée par :

Thanh Thuy Tien TA

soutenue le 6 juillet 2018

pour obtenir le grade de : Docteur de l'Université de Tours

Discipline / Spécialité : INFORMATIQUE

New single machine scheduling problems with deadlines for the characterization of optimal solutions

Thèse dirigée par :	
Billaut Jean-Charles	Professeur, Université de Tours
Rapporteurs :	
Lopez Pierre	DR CNRS, LAAS-CNRS, Toulouse
Chretienne Philippe	Professeur, Université Paris 6, Paris
Jury :	
Pinson Eric	Professeur, Université Catholique d'Angers
Soukhal Ameur	Professeur, Université de Tours

Résumé

Nous considérons un problème d'ordonnancement à une machine avec dates de fin impratives et nous cherchons caractériser l'ensemble des solutions optimales, sans les énumérer. Nous supposons que les travaux sont numérotés selon la règle EDD et que cette séquence est réalisable. La méthode consiste à utiliser le treillis des permutations et d'associer à la permutation maximale du treillis la séquence EDD. Afin de caractériser beaucoup de solutions, nous cherchons une séquence réalisable aussi loin que possible de cette séquence. La distance utilisée est le niveau de la séquence dans le treillis, qui doit être minimum (le plus bas possible). Cette nouvelle fonction objectif est étudiée. Quelques cas particuliers polynomiaux sont identifiés, mais la complexité du problème général reste ouverte. Quelques méthodes de résolution, polynomiales et exponentielles, sont proposées et évaluées. Le niveau de la séquence étant en rapport avec la position des travaux dans la séquence, de nouvelles fonctions objectifs en rapport avec les positions des travaux sont identifiées et étudiées. Le problème de la minimisation de la somme pondérée des positions des travaux est prouvé fortement NP-difficile. Quelques cas particuliers sont étudiés et des méthodes de résolution proposées et évaluées.

Mots clés : ordonnancement, une machine, dates impératives, positions, treillis, caractérisation, complexité

Abstract

We consider a single machine scheduling problem with deadlines and we want to characterise the set of optimal solutions, without enumerating them. We assume that jobs are numbered in EDD order and that this sequence is feasible. The key idea is to use the lattice of permutations and to associate to the supremum permutation the EDD sequence. In order to characterize a lot of solutions, we search for a feasible sequence, as far as possible to the supremum. The distance is the level of the sequence in the lattice, which has to be minimum. This new objective function is investigated. Some polynomially particular cases are identified, but the complexity of the general case problem remains open. Some resolution methods, polynomial and exponential, are proposed and evaluated. The level of the sequence being related to the positions of jobs in the sequence, new objective functions related to the jobs positions are identified and studied. The problem of minimizing the total weighted positions of jobs is proved to be strongly NP-hard. Some particular cases are investigated, resolution methods are also proposed and evaluated.

Keywords : scheduling, single machine, deadlines, positions, lattice, characterization, complexity

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Making a schedule and following a schedule is an ancient human activity, which is encountered in every aspect of life. About more than 100 years ago, Henry Laurence Gantt, who is best known for his work in the development of scientific management, created the famous "Gantt chart", which is the chart that illustrates a project schedule. This chart is one of the most famous basic knowledge, which leads to Advanced Planning and Scheduling systems that rely on sophisticated algorithms. Some of the first publications appeared in the Naval Research Logistics Quarterly in the mid fifties and contained the results by W.E. Smith [Smith, 1956], S.M. Johnson, [Johnson, 1954] and J.R. Jackson [Jackson, 1955]. After that period, because the scheduling problems become closer and closer to industrial applications, it increases the *complexity* and there has been a growing interest in *scheduling*.

Since this period, together with the development of the complexity theory [Cook, 1971], the scheduling problems have been intensively investigated, both for their possible applications and for their interest from a theoretical point of view. A classification of the problems has been standardized, with respect to their complexity.

In a very large majority of studies, the authors consider a scheduling problem, and (1) establish the complexity of the problem, and/or (2) propose resolution algorithms to find a solution to the problem (optimal or as close as possible to the optimal one, with or without performance guaranty, etc.).

In this thesis, we consider a very well known problem of the scheduling literature, and we search for the characteristics of the set of optimal solutions, without enumerating them. To our point of view, it is an original research topic, for which very few results have been found up to now.

This chapter introduces the basic concepts and components of scheduling theory and all the elements required to understand the contribution of this thesis. The outline of the thesis is given at the end of the chapter.

1.1 Introduction to the context of the study - required background

This study takes place in the field of *scheduling theory*. Before introducing some notions in scheduling theory, we introduce basic notions in the field of *complexity theory*.

1.1.1 Complexity of algorithms

Algorithmic complexity is concerned about how fast or slow a given algorithm performs. Algorithm complexity is a numerical function denoted by $T(n)$ which gives the computation time versus the input size n of the algorithm, without considering implementation details. Function $T(n)$ is the number of elementary steps performed by the algorithm, assuming that the running time of one step is a constant.

The complexity of an algorithm depends on estimating its processing cost in time (time complexity) but also in the required space memory (spatial complexity). By default, we talk about the time complexity.

In order to classify the algorithms according to their performances, the time function $T(n)$ is restricted to its asymptotic notation, using the "big-O" notation. For example, an algorithm with complexity $T(n) = 4n + 3n^2$ has the notation $O(n^2)$, meaning that the algorithm has a quadratic time complexity. Figure 1.1 shows the evolution of the running time for the most classical complexity functions, depending on the input size.

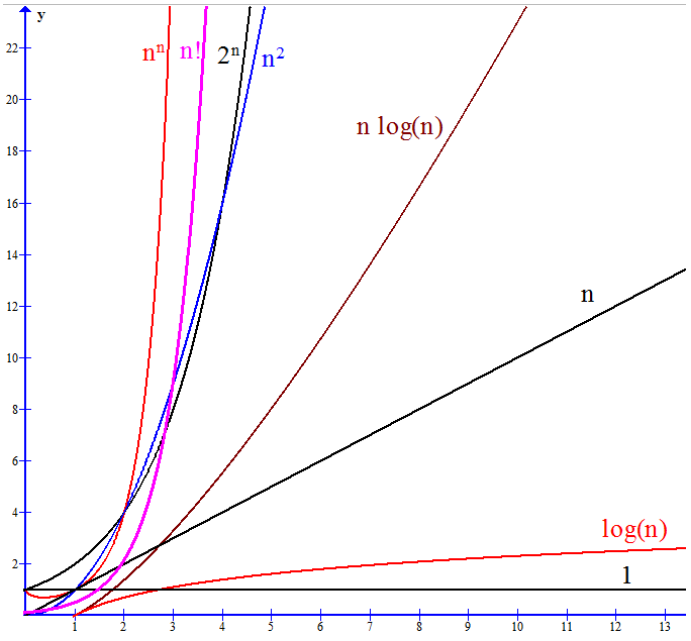

Figure 1.1: Computation times of common algorithm complexities

Table 1.1 shows some examples of common algorithms complexity.

Notice that in some cases, the complexity of an algorithm may be improved to the

Table 1.1: Common algorithms complexities

Name	$T(n)$	**Example**
Constant	$O(1)$	Determining if an integer (represented in binary) is even or odd
Logarithmic	$O(\log n)$	Binary search
Linear	$O(n)$	Finding the smallest or largest item in an unsorted array
Linearithmic	$O(n \log n)$	Fastest sorting algorithm
Quadratic	$O(n^2)$	Karmarkar's algorithm for linear programming; AKS primality test
Exponential	$2^{O(n)}$	Solving the traveling salesman problem using dynamic programming
Factorial	$O(n!)$	Solving the traveling salesman problem via brute-force search

detriment of the spatial complexity: it is possible to reduce the computational time by increasing the size of the stored data. However, such a step often leads to adding new functions, uniquely dedicated to the management of these data.

1.1.2 Introduction to Complexity theory

After Richard Karp's famous paper on complexity theory "Reducibility Among Combinatorial Problems" [Karp, 1972], the research in the 1970s focused mainly on the complexity of scheduling problems. We can refer the famous book "Computers and Intractability" of Michael R. Garey and David S. Johnson [Garey et Johnson, 1990] to obtain "A Guide to the Theory of NP-Completeness".

1.1.2.1 Complexity of problems

Complexity theory proposes a set of results and methods to evaluate the intrinsic difficulty of solving problems. A problem belongs to a *class of complexity*, which informs us about the complexity of the "best algorithm" able to solve it. A complexity class is a set of problems of related complexity.

In mathematics and computer science, several types of problems can be distinguished by the following two main classes of problems:

- *Decision problems* that are defined by a name, an instance, which is a description of all the parameters, and a question for which the answer belongs to $\{yes, no\}$,
- and *Optimization problems* that are defined by a name, an instance and in which the aim is to find the best solution (with minimum or maximum) value of a given function.

a. Complexity of decision problems We denote by $\mathcal{P}$, the *class of all decision problems* which are *polynomially solvable*, i.e. for which the answer '*yes*' or '*no*' can be

found by an algorithm for which the complexity is bounded by a polynomial of n.

We denote by $\mathcal{NP}$ the class of decision problems for which the answer can be determined by a *non-deterministic Turing machine in polynomial time (or less)*. Or, equivalently, those decision problems for which an answer '*yes*' can be **checked** in polynomial time.

Definition 1 ***Reduction between problems***

A decision problem P_1 polynomially reduces to a decision problem P_2 if and only if there exists a polynomial time algorithm f, which can build, from any instance I_1 of P_1, an instance $I_2 = f(I_1)$ of P_2 such that the response to problem P_1 for instance I_1 is 'yes' if and only if the answer to problem P_2 for instance I_2 is 'yes'.

If such an algorithm f exists, it proves that any instance of problem P_1 can be solved by an algorithm for problem P_2. We say that P_2 is at least as difficult as P_1.

If a polynomial time algorithm exists for solving P_2, then P_1 can also be solved in polynomial time.

A decision problem P_1 which polynomially reduces to a decision problem P_2 is denoted by $P_1 \propto P_2$.

Reductions are of course useful for optimization problems as well.

The next definition introduces an important subclass of the class $\mathcal{NP}$.

Definition 2 ***NP-completeness***

A problem P is NP-complete if P belongs to $\mathcal{NP}$ and any problem of $\mathcal{NP}$ polynomially reduces to P.

Lemma 1 *Let P, Q be decision problems. If $P \propto Q$, then $Q \in \mathcal{P}$ implies $P \in \mathcal{P}$ (and, equivalently, $P \notin \mathcal{P}$ implies $Q \notin \mathcal{P}$).*

Lemma 2 *Let P, Q, R be decision problems. If $P \propto Q$ and $Q \propto R$, then $P \propto R$.*

Note: If an NP-complete problem Q could be solved in polynomial time, then due to Lemma 1, all problems in $\mathcal{NP}$ could be solved in polynomial time and we would have $\mathcal{P} = \mathcal{NP}$.

Lemma 3 *If $P, Q \in \mathcal{NP}$, P is NP-complete, and $P \propto Q$, then Q is NP-complete.*

The class of *NP-complete problems* can also be divided into two parts. A problem P is NP-complete in the strong sense if P cannot be solved by a *pseudo-polynomial time algorithm*, unless $\mathcal{P} = NP$.

The NP-complete problems that can be solved by a pseudo-polynomial time algorithm are said to be *NP-complete in the ordinary sense*. We can refer to a more detailed discussion about ***strong and weak*** *NP-complexity* in [Garey et Johnson, 1990] and [Papadimitriou, 1994].

There are many NP-complete problems, the first problem proven to be *NP-complete* was SATISFIABILITY problem [Cook, 1971]. Some of the most important NP-complete problems that we use in this thesis are PARTITION and 3-PARTITION.

PARTITION problem	
Instance:	A finite set $A = \{a_i\}_n$ and a "size" $s(a) \in Z^+$ for each $a \in A$.
Question:	Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$?

The Partition problem is very important for scheduling theory. It is particularly useful for proving NP-Completeness results for problems involving numerical parameters, such as lengths, weights, costs, capacities, etc.

3-PARTITION problem	
Instance:	$A = \{a_i\}_{1 \leq i \leq 3m}$ a set of $3m$ elements such that $B/4 < a_i < B/2$ and $\sum_A a_i = mB$, with a bound $B \in N$.
Question:	Can A be partition into m disjont sets $A_1, A_2, ..., A_m$ such that $\sum_{A_k} a_i = B, \forall k \in \{1, 2, ..., m\}$ (note that each A_k must contain exactly 3 elements of A)

b. Complexity of optimization problems To any optimization problem, it is possible to associate a decision problem, by introducing a bound K, and asking to the existence of a solution with a cost smaller or greater than K (depending if the objective function is a min or a max). If the cost function is not difficult to compute, then the decision problem is not harder than the optimization problem. We say that if the decision problem is NP-complete, then the corresponding optimization problem is NP-hard.

NP-Hardness is a class of decision problems which are *at least as hard as the hardest problems in* $\mathcal{NP}$ (member of $\mathcal{NP}$ or not, may not even be decidable). NP-Hardness has the property that it *cannot be solved in polynomial time*, unless $\mathcal{P} = \mathcal{NP}$.

The following table illustrates the possible algorithms for solving NP-hard problems to optimality.

Complexity class	Algorithm	Example
Strongly NP-Hard	Exponential	2^n, $n!$, 3^n
Weakly or ordinary NP-Hard	Pseudo-polynomial	nW, $P.n^2$
P	Polynomial	n^2, $n \log n$

1.1.3 Required background in resolution methods

The goal of an optimization method is to find an optimal or near-optimal solution with low computational effort. The effort of an optimization method can be measured as the time (computation time) and space (computer memory) that is consumed by the method. There are different types of optimization methods to solve an optimization problem, with different efficiency.

Optimization methods can be roughly divided into two main categories: Exact methods (see 1.1.3.1) and Approximate (Heuristic and Metaheuristic) (see 1.1.3.2). *Exact methods* are guaranteed to find the optimal solution of the problem and to prove its optimality, for every finite size instance and with an instance dependent running time. *Approximate methods* do not have this guarantee and therefore generally return solutions that are worse than an optimal solution. However, for very difficult optimization problems (NP-hard or global optimization), the running time of exact methods may increase exponentially with respect to the dimensions of the problem, while heuristic methods usually find "acceptable" solutions in a "reasonable" amount of time. However, many heuristic methods are very specific and problem-dependent. So, we need the development of heuristic which are more general, called *metaheuristic* methods. Figure 1.2 presents the resolution methods in a graphical way.

In this section, we will not go into the details, and our presentation should only fulfill the needs of this thesis. We do not present examples, since they will be given in the later chapters.

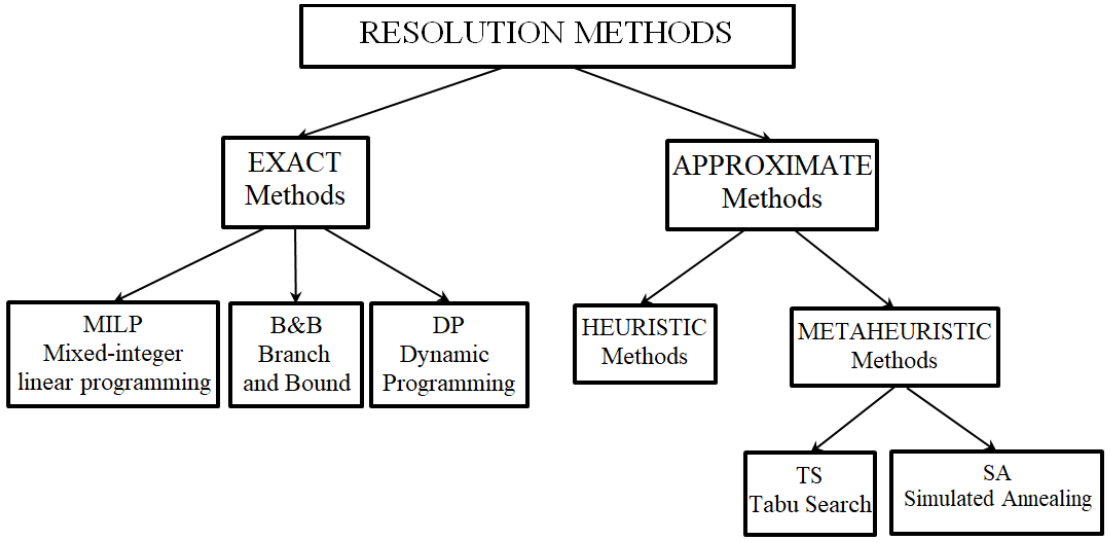

Figure 1.2: Resolution methods used in this thesis

1.1.3.1 Exact methods

In Computer Science and Operations Research, *exact algorithms* are algorithms that always solve an optimization problem to optimality. For the NP-hard problems, unless $\mathcal{P} = \mathcal{NP}$, such an algorithm cannot run in worst-case polynomial time, but there has been extensive research on finding exact algorithms whose running time are exponential with a low base.

Some exact resolution methods that we often meet are enumerative, such as *Dynamic Programming*, *Branch-and-Bound* (developed from Tree Search). *Integer Linear Programming* (ILP) or *Mixed Integer Linear Programming* (MILP) are based on the use of commercial solvers (or non commercial) which implement very sophisticated branching methods.

We describe three basic ways of formulating a scheduling problem using mathematical programming.

• **Modeling with mathematical programming**

Mixed Integer Linear Programming (MILP) is a very general framework for capturing problems with both discrete and continuous decision variables. MILP is a phase of modelisation of a problem under the following matrix form [Bixby *et al.*, 2000] and then, a phase of resolution of this problem by using a solver.

$$\begin{aligned} &\text{Minimize } c^T x \\ &\text{subject to } Ax \geq b \\ &\qquad\qquad l \leq x \leq u \end{aligned}$$

where some or all x_j variables are integer or binary, c are the cost parameters, A and b are the constraints parameters.

The problem is to find a feasible solution which minimizes the objective function.

A vector $x = (x_1, ..., x_n)$ satisfying the constraints is called a feasible solution.

A linear program that has a feasible solution is called **feasible**. A linear program may also be **unbounded**, i.e. for each real number K there exists a feasible solution x with $z(x) < K$. *Linear programs which have a feasible solution and are not unbounded always have an optimal solution.*

In scheduling, there are different ways to define the variables. We present now the most frequent definitions.

Positional Variables For these models, some binary variables represent the positions of jobs in a sequence. More precisely, we define binary variables

$$x_{j,k} = \begin{cases} 1 & \text{if job } J_j \text{ is in position } k \\ 0 & \text{otherwise} \end{cases} , \forall j \in \{1, 2, ..., n\}, \forall k \in \{1, 2, ..., n\}$$

This type of variables can be used when the considered scheduling problem is equivalent to finding a sequence of jobs. This sort of model has been introduced in [Wagner, 1959].

The following constraints ensure that there is exactly one job per position and one position per job:

$$\sum_{j=1}^{n} x_{j,k} = 1, \forall k \in \{1, 2, ...n\} \tag{1.1}$$

$$\sum_{k=1}^{n} x_{j,k} = 1, \forall j \in \{1, 2, ...n\} \tag{1.2}$$

Linear Ordering Variables The binary variables represent the relative positions of jobs. We define the following binary variables:

$$y_{i,j} = \begin{cases} 1 & \text{if job } J_i \text{ precedes job } J_j \\ 0 & \text{otherwise} \end{cases} , \forall i \in \{1, 2, ..., n\}, \forall j \in \{1, 2, ..., n\}$$

This type of model has been introduced in [Manne, 1960] for the job shop scheduling problem.

Time-Indexed Variables There are several possible definitions for time-indexed variables (see the initial paper [Bowman, 1959]). We define the following binary variables:

$$z_{j,t} = \begin{cases} 1 & \text{if job } J_j \text{ is being performed at time } t \\ 0 & \text{otherwise} \end{cases} , \forall j \in \{1, 2, ..., n\}, \forall t \in \{1, 2, ..., T\}$$

T is an upper bound to the maximum completion time. This quantity may be too big in the models. It is instance dependent. Another possibility is to say that it is equal to 1 if job j starts its processing at time t.

The disjunctive constraint is simply given by

$$\sum_{j=1}^{n} z_{j,t} \leq 1, , \forall t \in \{1, 2, ..., T\}$$

• Branch and bound (B&B)

The method was first proposed in [Land et Doig, 1960] for discrete programming problems. The name "Branch-and-Bound" first occurred in the work of [Little *et al.*, 1963] on the traveling salesman problem and in [Balas, 1983]. B&B is a problem-solving technique which is widely used for various problems in Operations Research.

The process of solving a problem using B&B algorithm can be described by a search tree. Each node of the search tree corresponds to a subset of feasible solutions to a problem. We assume in the following that the B&B algorithm is to find for a minimum value of a function $f(x)$, where x ranges over some set S of candidate solutions [Mehlhorn et Sanders, 2008], [Agnetis *et al.*, 2014].

The characteristics of this method are the following.

- *A branching rule*, that defines partitions of the set of feasible solutions into subsets. From a set S of feasible solutions, the branching returns two or more smaller sets S_1, S_2, etc. whose union covers S. The minimum of $f(x)$ over S is the minimum of the value v_1, v_2, etc. where each v_i is the minimum of $f(x)$ within S_i.
- *A lower bounding rule* that provides a lower bound $LB(S)$ on the value of the feasible solutions for any S. A good lower bound (with the highest possible value) may lead to the elimination of an important number of nodes of the search tree, but if its computational requirements are excessively large, it may become advantageous to use a weaker but more quickly computable lower bound.
- *A search strategy*, which selects the next node to explore. There are three basic search strategies: $depth-first$ (the list of nodes is managed as a LIFO list), $breadth-first$ (the list of nodes is managed as a FIFO list) and $best-first$ (nodes are sorted according to a sorting rule, generally the lower bound value is used).

- An *upper bounding method* UB of the objective value. The objective value of any feasible solution will provide such an upper bound for a minimization problem.

If the lower bound $LB(S_i)$ of a subproblem S_i is greater than or equal to UB, then this subproblem cannot yield to a better solution and there is no need to continue the branching from this node. We say that we *cut* this node. To stop the branching process in many nodes, the upper bound UB has to be as small as possible. Therefore, at the beginning of the B&B algorithm, some heuristic algorithms may be applied to find a good feasible solution with a small value of the objective function.

When the considered node is a leaf of the tree, it corresponds to a feasible solution. If the value of this solution is better than UB, then UB is updated and this solution is memorized.

The algorithm stops when the list of nodes to explore is empty.

A general formulation algorithm is given in Algorithm B&B 1.

Algorithm 1 B&B algorithm

```
1: UB = f(x_h): value of a heuristic solution x_h (if no heuristic is available, UB = ∞)
2: Initialize a list Q to hold a partial solution, with no variable assigned.
3: while Q ≠ ∅ do
4:     Take a node N of Q.
5:     if (N represents a single candidate solution x and f(x) < UB) then
6:         x is the best solution so far. Record it and UB ← f(x)
7:     else branch on N to produce new nodes N_i.
8:         for each child node N_i do
9:             if LB(N_i) < UB then store N_i on Q
10: return UB, x
```

• Dynamic Programming (DP)

Dynamic programming was invented by Richard Bellman [Bellman, 1957] and developed by the time. Depending on the subject, there are many ways to describe DP. Even only in computer science, we still have a lot of ways to present the concept of DP. With this thesis, we refer to the presentations given in [Blazewicz *et al.*, 2007] and [Agnetis *et al.*, 2014].

DP is a complete enumeration method. It decomposes recursively the problem into a series of subproblems, in order to minimize the amount of computation to be done. The approach solves each subproblem and determines its contribution to the objective function. At each iteration, it determines the optimal solution for a subproblem. The solution for the original problem can be deduced from the solution of each subproblem. Depending on the number of states and phases, the running time of a DP algorithm can be polynomial, pseudopolynomial or exponential. A DP formulation is characterized by three types of expressions:

- some initial conditions,

- a recursive relation,
- and a goal (i.e. an optimal value function).

1.1.3.2 Approximate methods

For approximate methods, too many methods are used in scheduling optimization, such as the applications of relaxations and the duality of MIP, heuristics, metaheuristics, matheuristics, etc. In this section, we limit the presentation to heuristics and metaheuristics. The applications of relaxations of MIP is not presented here even it is used for finding the lower bound in some parts.

It is also well known that there are a lot of contributors of heuristic methods in most of the aspect of optimization problems. We can say that heuristics are problem-dependent techniques "*(consisting of a rule or a set of rules) which seeks (and hopefully finds) good solutions at a reasonable computational cost. A heuristic is approximate in the sense that it provides (hopefully) a good solution for relatively little effort, but it does not guarantee optimality*" [Maniezzo *et al.*, 2009]. However, they usually get trapped in a local optimum and thus fail, in general, to obtain the global optimum solution.

Meta-heuristics, on the other hand, are problem-independent techniques. There are so many formal definitions and characteristics based on a variety of definitions from different authors derived from several books and a lot of survey papers have been published such as [Voss *et al.*, 1998], [Glover et Kochenberger, 2003], [Ólafsson, 2006], [Dréo *et al.*, 2006], [Brucker, 2007], [Maniezzo *et al.*, 2009], [Boussaid *et al.*, 2013], [Gonçalves *et al.*, 2016].

For computer science and mathematical optimization, Metaheuristic can be considered as a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity. Especially, they do not take advantage of the specificity of any problem, and so they may be used for a variety of problems. Some well known metaheuristic methods are Simulated Annealing (SA), Tabu Search(TS), Evolutionary Algorithms (genetic algorithms GA , evolutionary strategies, evolutionary programs, scatter search, and memetic algorithms), Cross Entropy Method, Ant Colony Optimization (ACO), Corridor Method (CM), Pilot Method, Adaptive memory programming (AMP), etc.

In this thesis, we consider some heuristic methods and two main metaheuristics methods: Simulated Annealing (SA) and Tabu Search (TS), described below.

• Simulated Annealing

Simulated Annealing (SA) as it is known today is motivated by an analogy to annealing in metallurgy. The idea of SA comes from [Metropolis *et al.*, 1953].

Annealing is a physical process. When we heat a solid past melting point and then cool it, the structural properties of the solid depend on the rate of cooling. If the liquid is cooled slowly enough, large crystals will be formed. However, if the liquid is cooled quickly (quenched) the crystals will contain imperfections. Metropolis's algorithm simulated the material as a system of particles. The algorithm simulates the cooling process

by gradually lowering the temperature of the system until it converges to a steady, *frozen* state [Dréo *et al.*, 2006].

In 1983, [Kirkpatrick *et al.*, 1983] took the idea of the Metropolis algorithm and applied it to optimization problems. The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution. Some interesting theory with applications can refer in [Chibante, 2010].

The SA algorithm requires the definition of an initial solution, an initial temperature, a perturbation mechanism, an objective function, a cooling schedule, and a terminating criterion.

Initial solutions: Some algorithms require the use of several initial solutions, but it is not the case of SA.

Initial temperature: The control parameter T must be carefully defined since it controls the acceptance rule defined by $e^{-\Delta/T}$: T must be large enough to enable the algorithm to move off a local minimum but small enough not to move off a global minimum.

Perturbation mechanism: it corresponds to the definition of the neighborhood.

Objective function: denoted $f(S)$, with S a feasible solution.

Cooling schedule: The cooling schedule is a rule to define the temperature variation. For example: "update $T_{i+1} = aT_i$, $0 < a < 1$ after each iteration".

Terminating criterion: There are several methods to control the termination of the algorithm. Some examples are:

- maximum number of iterations,
- minimum temperature value,
- minimum value of the objective function. The algorithm stops when the best objective function value is less than or equal to a given value.
- minimum value of acceptance rate,
- maximum computation time.

Fig. 1.3 of "Ball on terrain" is the illustration for a minimization problem solved by SA.

A general formulation algorithm is given in Algorithm SA 2.

A disadvantage of AS is that it may visit several times the same solutions. However, this method doesn't cost much of memory to remember the set of solutions that have been visited. The Tabu Search (which we are going to present in the following) has not this disadvantage, but it requires more memory.

• Tabu Search

Tabu search is a metaheuristic method proposed by Fred W. Glover in [Glover, 1986] and [Glover, 1989] for mathematical optimization, based on local search. We can also refer to [Glover et Laguna, 1997], [Rego et Alidaee, 2005], or [Gendreau et Potvin, 2010].

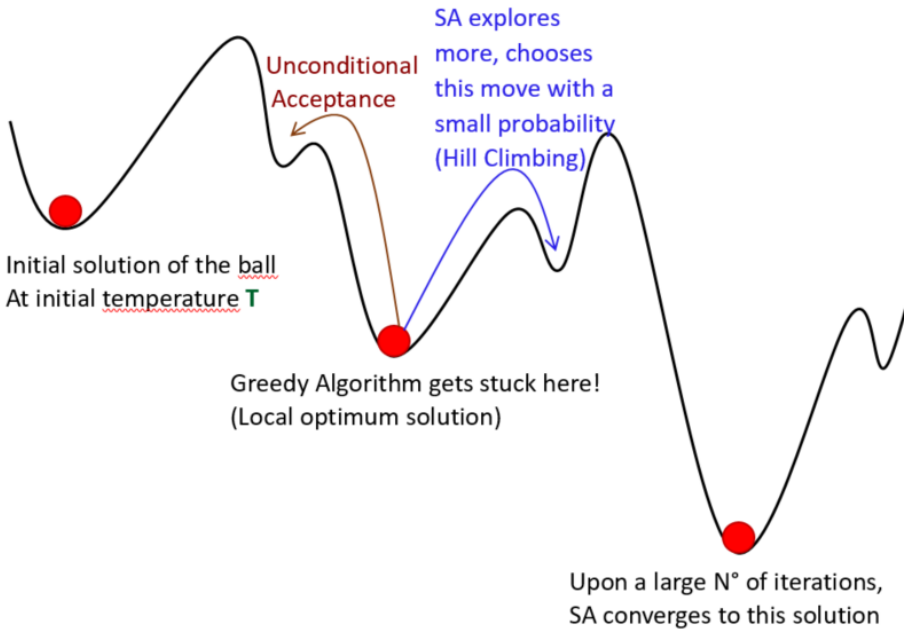

Figure 1.3: Illustration for a minimization problem with Simulated Annealing

Algorithm 2 SA algorithm

$i = 0$
$S = \{S_0\}$, initial solution, $S^* = S_0$, $best = f(S_0)$
$T = T_0$, initial temperature
while Terminating criterion is not satisfied **do**
    $S' =$ neighboring solution of S
    if $f(S') < best$ **then** $S^* = S', best = f(S')$
    else
        if $random[0,1] < \min\left\{1, e^{\frac{f(S)-f(S')}{T}}\right\}$ **then** $S = S'$
    $T_{i+1} = g(T_i)$
    $i = i + 1$
return S^*

Tabu search has a mechanism to help get out of local minima. Indeed, if a solution has been already visited within a certain short-term period, it is marked as "tabu" (i.e. forbidden), so that the algorithm will not consider that solution again in the future. So, the method costs some memory structures that describe the previously visited solutions.

For certain problems, the Tabu method is known to give excellent results; moreover, in its basic form, the method requires less parameters than Simulated Annealing, which makes it easier to use. However, the various additional mechanisms, like the intensification and diversification, increase its complexity and at the end, the setting of a lot of parameters is needed.

Some conceptions and definitions between SA and TS are the same (initial solution, perturbation mechanism, objective function, terminating criterion). We only present some details on the basic concepts of TS which differ from SA: the Tabu list and the Tabu size.

- Tabu list: it records the recent history of the search, which is stored in a short-term memory (a fixed and fairly limited quantity of information is recorded). In any context, there are several possibilities regarding the specific information that is recorded. We list some ways below:
  - *record complete solutions*, which needs a lot of storage and makes it expensive to check whether a potential move is tabu or not (it is therefore seldom used),
  - *record the last few transformations* performed on the current solution, prohibiting reverse transformations, this is the most commonly used possibility,
  - *record based on the key characteristics of the solutions* themselves or of the moves.
  - *record by Multiple tabu lists* can be used simultaneously and are sometimes interesting. For example, when different types of moves are used to generate the neighborhood, it might be a good idea to keep a separate tabu list for each type. Standard tabu lists are usually implemented as circular lists of fixed length.
- Tabu size: it has been shown that fixed-length tabu algorithms cannot always prevent cycling, and some authors have proposed varying the tabu list length during the search. Its setting generally comes from empirical experiments.

Despite the use of a Tabu list, the iterative improvement may stop at local optima, which can be very "poor". Some techniques and strategies required to prevent the search from getting trapped and to escape from them.

To escape from local optima, we may need to consider *local intensification* and *global diversification* mechanisms. They are the driving forces of metaheuristic search but there is no common rule to explain how to balance these two important components.

- Intensification (exploitation) improves solutions by exploiting the accumulated search experience (e.g., concentrating the search in a small search space area). In intensification, the promising regions are explored more thoroughly in the hope to find better solutions.

- Diversification (exploration) explorates "in the large" of the search space, trying to identify the regions with high quality solutions. In diversification, non-explored solutions must be visited to be sure that all regions of the search space are evenly explored and that the search is not confined to only a reduced number of regions.

Fig. 1.4 of "Ball on terrain" illustrates the minimization problem that is solved by TS.

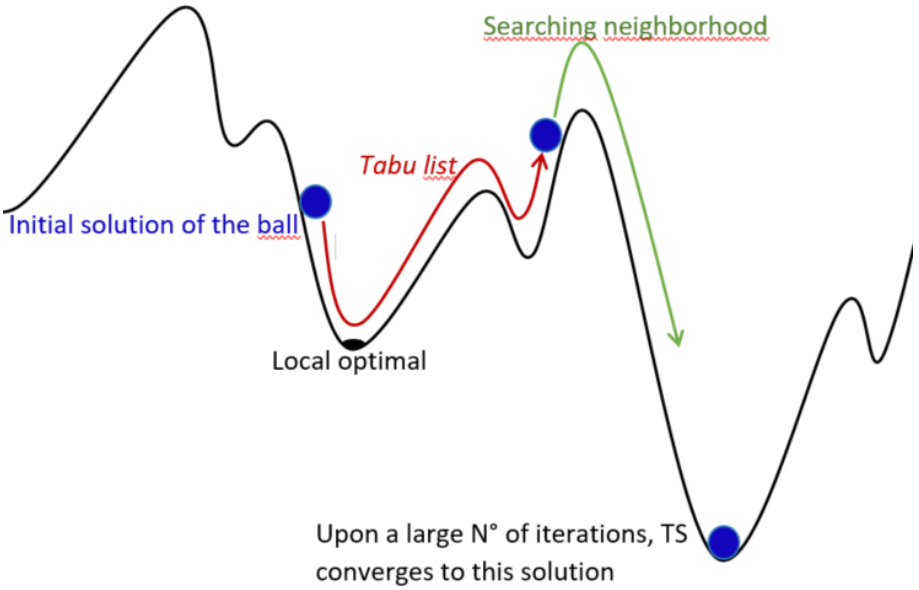

Figure 1.4: Illustration for a minimization problem with Tabu Search

The general TS algorithm 3 is described below. We have:

- S_0 an initial solution, S^* the best known solution, S the current solution,
- f is the value of solution S, f^* is the value of S^*,
- $N(S)$ is the whole neighborhood of S,
- $N'(S)$ is the neighborhood of S which is not tabu,
- T is the tabu list.

Algorithm 3 TS algorithm

1: $S = S_0$, initial solution
2: $S^* = S_0$
3: $f^* = f(S_0)$
4: Tabu list $T = \emptyset$
5: **while** Terminating criterion is not satisfied **do**
6: Select $S = min_{S' \in N'(S)} f(S')$, where S' is a neighbor of S,
7: **if** $(f(S) < f^*)$ **then** $f^* = f(S), S^* = S$
8: Record the current move in the tabu list T (delete oldest entry if necessary)
9: **return** S^*

Similarly to Alg. 2, this method can be improved by intensification and diversification mechanisms.

1.1.4 Introduction to Scheduling theory

We give in this section basic notions and notations in scheduling.

1.1.4.1 Definition

There are many definitions of scheduling problems. Scheduling problems are encountered in a lot of types of systems, when it is necessary to organize and/or distribute the work between some entities. We find in every book in the literature a definition of a scheduling problem, as well as its principal components. Among these definitions we quote the following one [Carlier et Chrétienne, 1988]:

"Scheduling is to forecast the processing of a work by assigning resources to tasks and fixing their starting times. [...] The different components of a scheduling problem are the tasks, the potential constraints, the resources and the objective function. [...] The tasks must be programmed to optimise a specific objective [...] Of course, often it will be more realistic in practice to consider several criteria."

A scheduling problem deals with *jobs* to schedule, each job may be broken down into a series of *operations*. The operations of a job may be connected by *precedence constraints*. The *resources or machines* can perform only one operation at a time. To solve a scheduling problem, we may have to solve an *assignment* problem. The purpose of scheduling is to **minimize** functions depending on jobs completion times and *costs*.

To simplify the problem description and to know if a scheduling problem is already treated in the literature, we use a notation for problems. The notation which is now admitted in the literature was introduced by [Graham *et al.*, 1979] (see a detailed description in [Blazewicz *et al.*, 2007]). This notation is divided into three fields: $\alpha|\beta|\gamma$.

- field α may break down into two fields: $\alpha = \alpha_1\alpha_2$. The values of α_1 and α_2 refer to the machines *environment* of the problem and possibly with the number of available machines,
- field β contains the explicit *constraints* of the problem to respect,
- field γ contains the *criterion/criteria* to optimize.

We can refer to a lot of books, definitions, propositions, notations, etc. Each book has an introduction to a particular field. For more details, the reader can refer to:

- [Brucker, 2007] for a global overview of scheduling algorithms,
- to [Blazewicz *et al.*, 2007] for another presentation of scheduling problems in computer and manufacturing processes,
- to [Pinedo, 2016] where supplementary material is included in the form of slide-shows from industry and movies, that show implementations of scheduling systems,

- to [T'Kindt et Billaut, 2006] for an overview of multicriteria scheduling problems,
- to [Jozefowska, 2007], dedicated to just-in-time scheduling problems,
- to [Agnetis *et al.*, 2014] which is dedicated to multiagent scheduling problems.

All these books give enough details on the main theories related to the context of our study.

1.1.4.2 Notations

A scheduling problem is characterized by a set $\mathcal{J}$ of n jobs. To each job J_j is associated a processing time denoted by p_j $(1 \leq j \leq n)$, a due date d_j or deadline (means strict due date) $\tilde{d}_j$. For some problems, a weight w_j is also associated to each job J_j $(1 \leq j \leq n)$.

The completion time of a job J_j is denoted by C_j. Its tardiness T_j is defined by $T_j = \max(0, C_j - d_j)$.

1.1.4.3 Shop environments

There are several classes for the shop environments.

a. Scheduling problems without assignment We distinguish:

- **Single machine** (1) : Only a single machine is available for the processing of jobs. It concerns a basic shop or a shop in which a unique machine poses a real scheduling problem. The case of a single machine is the simplest of all possible machine environments and is a special case of several more complicated machine environments.
- **Flowshop (Fm)**: m machines are available in the shop. The jobs use the machines in the same order, from the machine M_1 to the last machine M_m. We say that they have the same routing. In a *permutation flowshop* we find in addition that each machine has the same sequence of jobs: they cannot overtake each other.
- **Jobshop (J)**: several machines are available in the shop. Each job has its own routing.
- **Openshop (O)**: several machines are available in the shop. The jobs do not have fixed routings. They can, therefore, use the machines in any order.

b. Scheduling and assignment problems We assume that the machines can perform the same operations. The problem is two folds: assigning one machine to each operation and sequencing the operations on the machines. We can differentiate between the following configurations:

- **identical machines (P)**: an operation has the same processing time on any machine.

- **uniform machines (Q)**: the processing time of an operation depends on the number of components the machine can process per unit of time.

- **unrelated or independent machines (R)**: the processing time of an operation depends on the operation and on the machine.

1.1.4.4 Constraints

A solution of a scheduling problem must always satisfy a certain number of constraints, explicit or implicit. For example, in a single machine scheduling problem, it is implicit that the machine can perform only one job at a time. In this section we describe the explicit constraints encountered most frequently in scheduling.

- **pmtn** indicates that preemption is authorized: it is possible to interrupt a job and to resume its processing later, possibly on another resource.

- **prec** indicates that the operations are connected by precedence constraints. 'prec' leads to different cases according to the nature of the constraints: *prec* to describe the most general case, *tree*, *in-tree, outtree, chains* and *sp-graph* (for series-parallel graph ; see [Pinedo, 2016] or [Brucker, 2007] to denote some particular cases).

- **batch** indicates that the operations are grouped into batches. Two types of batch constraints are differentiated in the literature: the first called *s-batch* concerns serial batches where the operations constituting a batch are processed in sequence and the second called *p-batch* concerns parallel batches where the operations constituting a batch are processed in parallel on a cumulative resource. In both cases, the completion time of an operation is equal to the completion time of the batch. In the first case, the duration of the batch is equal to the sum of the processing times of the operations which constitute it, whereas in the second case its duration is equal to the longest processing time of the operations in the batch.

- $d_j = d$ indicates that all the due dates are identical. A due date may be not respected. In this case, the quality of the solution is generally given by a measure of the tardiness. Likewise $\widetilde{d_j} = \widetilde{d}$ for the deadlines. A deadline must be respected, and a part of the problem is to find a feasible solution.

- $p_j = p$ indicates that the processing times are all identical. We often encounter this constraint with $p = 1$.

1.1.4.5 Optimality criteria

We can classify the optimality criteria into two large families: "minimax" criteria, which represent the **maximum** value of a set of functions to be minimized, and "minisum" criteria, which represent a **sum** of functions to be minimized.

a. Minimisation of a maximum function: "minimax" criteria We define here the most important objective functions in scheduling.

- $C_{max} = \max_{j=1}^{n} C_j$, the makespan, with C_j the completion time of job J_j.
- $L_{max} = \max_{j=1}^{n} L_j$, the maximum lateness, with $L_j = C_j - d_j$ the lateness of J_j.
- $T_{max} = \max_{j=1}^{n} T_j$, the maximum tardiness, with $T_j = \max(0, C_j - d_j)$ the tardiness of J_j.

Generally, f_{max} will refer to an ordinary "minimax" criterion, which is a non decreasing function of the completion times of jobs.

b. Minimisation of a sum function: "minisum" criteria "Minisum" criteria are usually more difficult to optimize than "minimax" criteria. We write "$\sum$" for "$\sum_{j=1}^{n}$" when there is no ambiguity. Among the minisum criteria, we have:

- $\sum C_j$ or $\frac{1}{n}\sum C_j$. This criterion represents the average completion time or total completion time of jobs.
- $\sum w_j C_j$ or $\frac{1}{n}\sum w_j C_j$. This criterion represents the average weighted completion time or total weighted completion time of jobs.
- $\sum T_j$ or $\frac{1}{n}\sum T_j$. This criterion represents the average or total tardiness of jobs.
- $\sum w_j T_j$ or $\frac{1}{n}\sum w_j T_j$. This criterion represents the average weighted tardiness or total weighted tardiness of jobs.

In a general way, $\sum f_j$ designates an ordinary "minisum" criterion which is usually a non decreasing function of the jobs completion times.

1.1.5 Required background in single machine scheduling

We give in this section the basic notions in scheduling algorithms, required for a good understanding of the rest of the thesis.

1.1.5.1 Solving the $1||\sum C_j$ and $1||\sum w_j C_j$ problems

We consider a single machine scheduling problem. To each job is associated a processing time and a weight. The objective is to find a solution minimizing the sum of completion times, or the weighted sum of completion times.

Definition 3 *We define SPT (Shortest Processing Time first) a sorting rule, that sorts the jobs in a non decreasing order of the processing times. The converse rule is the rule LPT (Longest Processing Time first).*

Definition 4 *We define WSPT (Weighted Shortest Processing Time first) a sorting rule, that sorts the jobs in a non decreasing order of the ratio processing time divided by the weight.*

Proposition 1 *[Smith, 1956] Sequencing the jobs in SPT order gives an optimal sequence for the* $1||\sum C_j$ *problem. Therefore, the problem is in P, and can be solved in* $O(n \log n)$*. Sequencing the jobs in WSPT order gives an optimal sequence for the* $1||\sum w_j C_j$ *problem. Therefore, the problem is in P, and can be solved in* $O(n \log n)$*.*

1.1.5.2 Solving the $1||L_{\max}$ problem

We consider a single machine scheduling problem. To each job is associated a processing time and a due date. The objective is to find a solution minimizing the maximum lateness.

Definition 5 *We define EDD (Earliest Due Date first) a sorting rule, that sorts the jobs in a non decreasing order of the due dates.*

Proposition 2 *[Jackson, 1955] Sequencing the jobs in EDD order gives an optimal sequence for the* $1||L_{\max}$ *problem. Therefore, the problem is in P, and can be solved in* $O(n \log n)$*.*

We notice that this rule also solves the $1||T_{max}$ problem optimally.

We denote by L^*_{max} the value of the optimal maximum lateness. We define

$$\widetilde{d_j} = \min\left(d_j + L^*_{max}, \sum_{j=1}^{n} p_j\right)$$

for each job J_j. Because for this problem, a right-shifted schedule is always optimal, there is no reason to keep a deadline greater than the sum of processing times.

Finding a sequence respecting these deadlines is equivalent to find a sequence with an optimal maximum lateness.

Example 1 *Consider a 5-job instance with* $p = (6, 7, 2, 1, 10)$ *and* $d = (16, 31, 9, 12, 13)$*.*

The sequence given by EDD rule is $EDD = (J_3, J_4, J_5, J_1, J_2)$*. The completion times of the jobs (from* J_1 *to* J_5*) are equal to* $C = (19, 26, 2, 3, 13)$ *and the lateness of jobs are equal to* $T = (3, -5, -7, -9, 0)$*. Therefore, we have* $L^*_{max} = 3$*.*

Now, we renumber the jobs so that $EDD = (J_1, J_2, ..., J_5)$ *and we associate to each job a deadline* $\widetilde{d_j}$*. We obtain:*

	J_1	J_2	J_3	J_4	J_5
p_j	2	1	10	6	7
$\widetilde{d_j} = L^*_{max} + d_j$	12	15	16	19	26

Because finding a sequence respecting these deadlines is equivalent to find a sequence with an optimal maximum lateness, in the rest of the thesis, we will only consider the problem with the deadlines and jobs numbered in EDD order.

1.1.5.3 Solving the $1|prec|f_{max}$ problem

We consider the problem $1|prec|f_{max}$ with $f_{max} = max(f_j(C_j))$, f_j monotone for $\forall j \in \{1, 2, ..., n\}$. A set of precedence constraints $prec$ between jobs is defined and no preemption is allowed.

The algorithm builds an optimal sequence π [Lawler, 1973] in reverse order.

Let $N = \{1, 2, ..., n\}$ be the set of all jobs and $S \subseteq N$ the set of unscheduled jobs. We define $p(S) = \sum_{j \in S} p_j$. The scheduling rule may be formulated as follows: schedule a job $j \in S$, which has no successor in S and with a minimal value $f_j(p(S))$, as the last job in S.

To give a precise description of the algorithm, we represent the precedence constraints by the corresponding **adjacency matrix** $A = (a_{i,j})$ where $a_{i,j} = 1$ if and only if J_j is a direct successor of J_i. By $n(i)$ we denote the number of immediate successors of J_i.

The algorithm BW-Lawler is presented in Algorithm 4.

Algorithm 4 BW-Lawler for problem $1|prec|f_{max}$

```
1: for i = 1 to n do n(i) = Σ_{j=1}^{n} a_{i,j}
2: S = {J_1, J_2, ..., J_n}, t = Σ_{j=1}^{n} p_j
3: for k = n down to 1 do
4:     Find job J_j ∈ S with n(j) = 0 and minimal value f_j(t)
5:     S = S \ {J_j}; n_j = ∞; π(k) = J_j ; t = t − p_j
6:     for i = 1 to n do
7:         if a_{i,j} = 1 then n(i) = n(i) − 1
```

The complexity of this optimal algorithm is $O(n^2)$.

1.1.5.4 Solving the $1|\widetilde{d}_j|\sum w_j C_j$ problem

We consider a single machine scheduling problem. To each job is associated a processing time, a weight and a deadline, that has to be respected. The objective is to find a feasible solution (respecting the deadlines) and minimizing the total weighted completion times. This problem is NP-hard [Lenstra *et al.*, 1977].

We describe here the heuristic "Smith's backward scheduling rule". We define $t = \sum_{j=1}^{n} p_j$. Provided there exists a schedule in which all jobs meet their deadlines, the algorithm chooses one job with largest ratio p_j/w_j among all jobs J_j with $\widetilde{d}_j \geq P$, and schedules the selected job last. It then continues by choosing an element of best ratio among the remaining $n-1$ jobs and placing it in front of the already scheduled jobs, etc.

This algorithm BW-Smith is described in Alg. 5.

This algorithm can be implemented in $O(n \log n)$. We also know that the algorithm is exact in the following cases:

(i) unit processing times, i.e. for the problem $1|p_j = 1, \widetilde{d}_j|\sum w_j C_j$

Algorithm 5 BW-Smith for problem $1|\widetilde{d}_j|w_jC_j$

1: $S = \{J_1, J_2, ..., J_n\}$ a set of jobs, $t = \sum_{j=1}^{n} p_j$
2: **while** $S \neq \varnothing$ **do**
3: $\quad S_t = \{J_j \in S / \widetilde{d}_j \geq t\}$
4: $\quad$ Choose $J_j \in S_t$ such that p_j/w_j is minimal
5: $\quad$ Schedule J_j in position n
6: $\quad n = n - 1$, $S = S \setminus \{j\}$, $t = t - p_j$

(ii) unit weights, i.e. for problem $1|\widetilde{d}_j| \sum_{j=1}^{n} C_j$ [Smith, 1956]

(iii) agreeable weights, i.e. for problems where $p_i \leq p_j$ implies $w_i \leq w_j$, $\forall i, j \in \{1, 2, ...n\}$.

1.2 Characterization of solutions

As it has been explained earlier (see Section 1.1.4), the resolution of a scheduling problem consists in giving a starting time for each job. And solving a scheduling problem consists in returning one solution (one schedule), as best as possible, or possibly optimal.

It is well known that some problems are really hard to solve, and finding one optimal – or feasible – solution is really challenging. It is also well known that some scheduling problems have a lot of optimal solutions (potentially an exponential number).

Example 2 *Let consider the* $1|\tilde{d}_j|-$ *problem (find a schedule of jobs respecting the deadlines). This problem is known to be solvable in polynomial time by sorting the jobs in EDD order (see Section 1.1.5.2). Let consider an n-job instance where each job* J_j *has a processing time equal to* p_j *and the following deadlines:* $\tilde{d}_1 = p_1$, $\tilde{d}_j = P$, $\forall i \in \{2, ..., n\}$ *and* $P = \sum_{j=1}^{n} p_j$. *All the sequences such that job* J_1 *is in first position are feasible. Therefore, for this instance, there are* $(n-1)!$ *feasible solutions.*

In a dynamic environment (some new jobs arrive in real time in the system and one has to insert these jobs in the current schedule, and some events occur, making the solution no more usable), it can be interesting to have some flexibility or robustness in the current solution, in order to make the decision maker able to react to hazards or unexpected events. For doing this, it could be interesting to have not only one solution to the problem, but several solutions. Furthermore, if we know that there are an important set of optimal solutions for a given objective function, it may be interesting to introduce another objective function, in order to be more precise and to obtain a more interesting solution.

In such contexts, having a set of solutions may be of interest, but if this set contains an exponential number of solutions, this is no more interesting. In some cases, an important element is to know the *characteristics* of the solutions in this set.

Some preliminary studies concerning the search of the characteristics of some solutions (characteristics of the solutions, but not the list) have been conducted in this direction since several years.

We present a survey of these technics, and then the characterization that in used in this thesis.

1.2.1 Survey of characterization methods

We present briefly three methods: the groups of permutable operations 1.2.1.1, the partial order 1.2.1.2 and the pyramid structures 1.2.1.3.

1.2.1.1 Groups of permutable operations

"Groups of permutable operations" is a scheduling method that consists in scheduling groups of jobs, instead of jobs. In each group, it is supposed that the processing order is not fixed between the jobs.

This notion has been first developed in the LAAS-CNRS laboratory in Toulouse in the 80s [Thomas, 1980], [Le Gall, 1989], [Billaut, 1993], [Artigues, 1997], and extensions have been proposed later [Esswein, 2005], [Pinot, 2008].

Let consider a job shop environment. On each machine M_k, we define a sequence of v_k groups $g_{k,1}$, ..., g_{k,v_k}. So, the set of operations processed on machine M_k is equal to $\cup_{r=1}^{v_k} g_{k,r} = \{O_{i,j} | m_i = M_k\}$ and $\cap_{r=1}^{v_k} g_{k,r} = \emptyset$ on any machine M_k.

Example 3 *Let consider a job shop with two machines and 4 jobs, each composed by two operations. It is equivalent to consider 8 independent jobs where:*

- *precedence relations* $J_1 \prec J_2$, $J_3 \prec J_4$, $J_5 \prec J_6$, $J_7 \prec J_8$,
- *operations* J_1, J_3, J_6, J_8 *are processed on machine* M_1 *and* J_2, J_4, J_5, *and* J_7 *are processed on machine* M_2.

We assume that the jobs have identical processing times, release dates are equal to zero, and identical due dates.

Fig. 1.5 represents a sequence of groups of permutable operations, where there are four groups, each group with two jobs. So, there are $2^4 = 16$ *potential solutions characterized by this sequence of groups.*

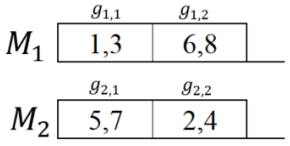

Figure 1.5: Groups of permutable operations

To evaluate a sequence of groups, two indicators can be given. One is the quality of the sequence in the worst case and one is the quality of the sequence in the best case.

Finding the best-case quality leads to an NP-hard problem that can be solved optimally using an exact method, because a group of permutable operations is a partial solution, for

which some decisions have not been taken. Finding the best-case quality is equivalent to consider the current groups as constraints, and to optimize the rest of the schedule. This is of course NP-hard in the case of a job shop problem.

Finding the worst-case quality can be done in polynomial time by applying the critical path method in a specific graph with activities on nodes.

Some indicators have been proposed, as for example the flexibility. The flexibility of a group sequence is related to the total number of groups, denote by Gps. A measure of flexibility is for example the number of characterized sequences, denoted by Seq. Thus, if Gps decreases (the number of groups decreases), then Seq increases and the quality of the worst characterized schedule decreases.

In reality, a compromise between the flexibility and the quality of the worst characterized solution can be searched. To solve this multicriteria scheduling problem, the authors use the $\varepsilon - constraint$ approach, assuming that the objective is to maximize the flexibility, respecting a threshold value of the quality.

In [Esswein *et al.*, 2005], the author tackles three classical two-machine shop problems: the $F2||C_{max}$, the $J2||C_{max}$ and the $O2||C_{max}$ problem with a measure for the flexibility defined by $\phi = \frac{2n-Gps}{2n-2}$.

1.2.1.2 Partial order between operations

In order to characterize a set of solutions S for a single machine problem where release times are associated to the operations, [Aloulou, 2002] and [Aloulou *et al.*, 2004] consider the *partial order between operations*. There are two criteria to measure the quality of a partial order. Because it is impossible to enumerate all the characterized solutions, only the quality of the best characterized solution and of the worst characterized solution for both criteria are determined. $Z_k^{min}(S)$ and $Z_k^{max}(S)$ denote the value of the best and the worst characterized solution, for criterion $Z_k, 1 \leq k \leq 2$.

The performance of a partial order between operations is a function of these four parameters plus the coordinates of the Utopian point M^* (minimum of each criterion when considered separately, denoted by Z_1^* and Z_2^*). Figure 1.6 illustrates these definitions for a set of solutions.

Besides, the author has an interesting measure of the performance of S.

$$D(S) = \alpha D_1(S) + (1-\alpha) D_2(S)$$

$$D_k(S) = \beta_k \frac{Z_k^{min}(S)}{Z_k^*} + (1-\beta_k)\frac{Z_k^{max}(S)}{Z_k^*}, \forall k, 1 \leq k \leq 2$$

where $\alpha, \beta_k \in [0,1]$ are real parameters. Clearly, the smaller the value $D(S)$, the better the solution S.

The ideal measure of the sequential flexibility is the number of characterized solutions, i.e. the number of "linear extensions of a partially ordered set". This problem is $\#P - complete$ as shown by [Brightwell et Winkler, 1991]. Thus, Aloulou [Aloulou, 2002] measures the sequential flexibility by the number of non-oriented edges in the transitive graph representing the partial order, i.e. the number of non fixed precedences, denoted

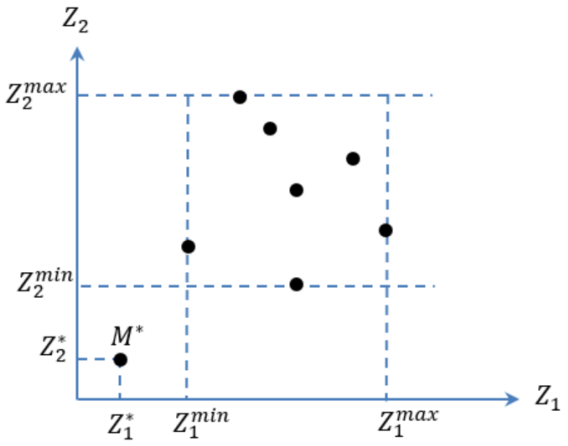

Figure 1.6: Quality of a set of solutions characterized by a partial order of operations

by $Z_{seqflex}$. As a measure for the temporal flexibility, the author proposes to compute the mean slack, where the slack is determined with the worst possible starting time for each operation, denoted by $Z_{tempflex}$. [Aloulou, 2002] provides a genetic algorithm to find solutions that minimize a linear combination of $D(S), Z_{seqflex}$ and $Z_{tempflex}$.

In [Policella *et al.*, 2005] and [Policella, 2005] the authors consider a Resource Constrained Project Scheduling Problem with minimum and maximum time lags. The aim is to propose a set of solutions with an implicit and compact representation of this set. The author introduces a *partial order schedule* (POS), i.e. a set of feasible solutions to a scheduling problem that can be represented by a *graph* with the activity on nodes and with arcs to represent the constraints between activities, such that any "time feasible" schedule defined by the graph is also a "feasible" schedule. The makespan of a POS is defined as the makespan of its earliest start schedule, where each activity is scheduled to start at its earliest start time. Some metrics are proposed to compare POS [Policella *et al.*, 2004]. Two metrics give an evaluation of the flexibility and one measure gives an evaluation of the stability of the solutions found. The first measure for evaluating the flexibility is $Z_{seqflex}$ (as defined in [Aloulou et Portmann, 2003]). The second metric is based on the slacks associated to the activities:

$$Z_{slackflex} = 100 \times \sum_{i \neq j} \frac{|d(C_j, t_i) - d(C_i, t_j)|}{H \times n \times (n-1)} = 100 \times \sum_{i \neq j} \frac{||C_j - t_i| - |C_i - t_j||}{H \times n \times (n-1)}$$

with t_i and C_i the starting time and the completion time of activity i, H the horizon time and n the number of activities.

This metric characterizes the fluidity of a solution, i.e. its availability to absorb temporal variation in the execution of activities. The higher the value of $Z_{slackflex}$ the higher the probability of localized changes.

To measure the stability, [Policella *et al.*, 2004] introduce a third measure, called disruptibility, denoted by Z_{disrup} and defined by:

$$Z_{disrup} = \frac{1}{n} \sum_{i=1}^{n} \frac{sl_i}{num(i, \Delta_i)}$$

where sl_i the slack of activity i (difference between latest and earliest starting times) and $num(i, \Delta_i)$ a function that returns the number of activities that are shifted in the process if activity i is shifted to the right for Δ_i time units. For solving the problem, [Policella *et al.*, 2004] also propose several heuristic algorithms.

1.2.1.3 Interval structures-pyramid

By using Allen's Interval Algebra [Allen, 1983] to describe the relative position of spatial objects, the interval relationships are analyzed and a sufficient optimality condition is established providing a characterization of a large subset of optimal sequences. There are many ways to describe the Allen's relations (figure 1.7), and to define the basic concepts of interval structure, we can refer a way below.

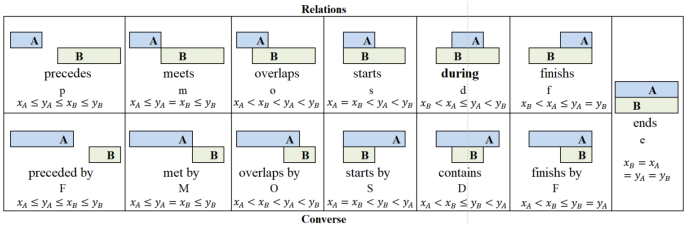

Figure 1.7: Allen's thirteen basic relations

La in [La, 2005] considers the problem $1|r_i, d_i|L_{max}$ and aims at proposing a set of solutions to the decision maker. An interval $[r_i, d_i]$ is associated to each operation O_i and an interval structure is defined based on Allen's relations.

Consider two intervals A, B, $during(A, B) = true$ if and only if $r_B < r_A \le d_A < d_B$. A top of an interval structure is an interval T such that for all A, Allen's relation $during(A, T)$ never holds. Given a top T_α, a $T - pyramid$ P_α is a set of intervals A such that $during(T, A)$ holds. [Erschler *et al.*, 1983] show that a set of dominant sequences is composed by sequences such that:

- the tops of intervals are sequenced in the r_i increasing order (d_i in case of equality),
- before the first top of interval, are sequenced the operations that belong to the first $T - pyramid$ in the r_i non decreasing order,
- after the last top of interval, are sequenced the operations that belong to the last $T - pyramid$ in the d_i non decreasing order,
- between two tops T_k and T_{k+1} are sequenced first the operations that belong to P_k and not to P_{k+1} in the d_i non decreasing order ; then the operations that belong to $P_k \cap P_{k+1}$ in an arbitrary order ; and finally the operations that belong to P_{k+1} but not to P_k in the r_i non decreasing order.

Example 4 *Figure 1.8 illustrates an interval structure for the following example:* $n = 6$ *jobs, release dates* $r = (2, 1, 0, 6, 4, 8)$*, due dates* $d = (3, 5, 10, 7, 9, 11)$*. The tops of this interval structure are operations* $T_i = 1$*,* $T_2 = 4$*,* $T_3 = 6$*, and the* $T - pyramids$

are $P_1 = \{2,3\}$, $P_2 = \{3,5\}$, $P_3 = \emptyset$. *This interval structure characterizes the following sequences: (3,2,1,5,4,6), (3,2,1,4,5,6), (3,1,2,5,4,6), (3,1,2,4,5,6), (2,1,3,5,4,6), (2,1,3,4,5,6), (1,2,3,5,4,6), (1,2,3,4,5,6), (2,1,5,4,3,6), (2,1,4,5,3,6), (1,2,5,4,3,6), and (1,2,4,5,3,6).*

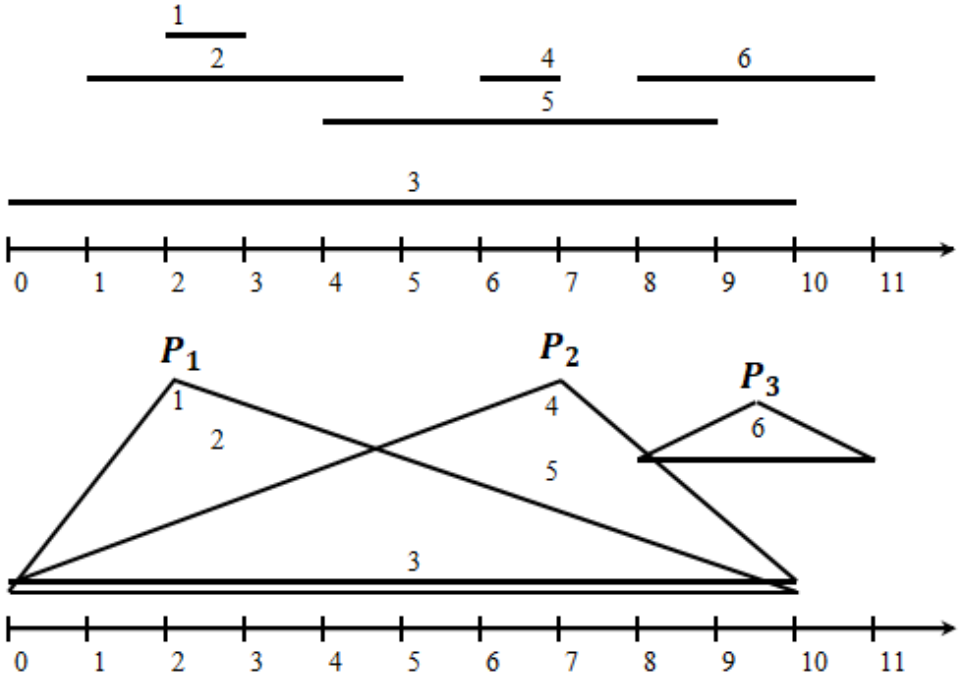

Figure 1.8: Illustration of an interval structure

[La, 2005] proposes one measure for the flexibility and two measures for the quality. The flexibility is equal to the number of characterized sequences, which is equal to $\prod_{q=1}^{P}(q+1)^{n_q}$, with P the number of pyramids and n_q the number of operations different from a top, that belong exactly to q pyramids.

For the example, $\prod_{q=1}^{P}(q+1)^{n_q} = (1+1)^2 \times (2+l)^1 \times (3+1)^0 = 4 \times 3 \times 1 = 12$ sequences.

Two measures are associated to each operation: its best possible lateness and its worst possible lateness, both computed in $O(n \log n)$. The quality of a set of solutions is measured by the maximum of the best possible lateness for all the operations, denoted by $max(L_j^{min})$ and by the worst possible lateness, denoted by $max(L_j^{max})$.

A branch-and-bound algorithm is proposed to characterize a set of solutions. Problems with 100 operations are solved in less than two seconds on the average, and enable to characterize up to 10^{12} sequences.

A base of an interval structure is an interval B such that for all A, Allen's relation $during(B, A)$ never holds. Given a base B_α, a $b - pyramid$ P_a related to B_α is the set

of intervals A such that $during(A, B)_\alpha$ holds. [Briand *et al.*, 2005] use the concept of $b-pyramid$ to characterize a subset of optimal sequences for the $F2||C_{max}$ problem. Two interval structures are defined: an interval structure with the jobs such that $p_{i,1} \leq p_{i,2}$, an interval $[p_{i,1}, p_{i,2}]$ is associated to these jobs; and an interval structure with the jobs such that $p_{i,1} \geq p_{i,2}$, an interval $[p_{i,2}, p_{i,1}]$ is associated to these jobs. A huge number of solutions are characterized in polynomial time, including all the Johnson's sequences.

This characterization can be considered as robust since the interval structures do not change if the relative order of the processing times $p_{i,2}, p_{i,1}$ remains unchanged.

To continue with the theory interval structures and the pyramids, the authors develop a robust approach for *single machine scheduling problem* $1|r_i|L_{max}$ in [Briand *et al.*, 2007].

1.2.2 A new way to characterize solutions

In this section, we present a new way to characterize the solutions. Some preliminary studies concerning the search of the characteristics of the optimal solutions have been conducted by using *the lattice of permutations* as support.

1.2.2.1 The lattice of permutation and properties

A lattice consists of a partially ordered set in which every two elements have a unique supremum (also called a least upper bound or join) and a unique infimum (also called a greatest lower bound or meet).

The lattice of permutations This paragraph is taken from [Billaut *et al.*, 2012].

Consider the set $\{1, 2, ..., n\}$ of integers and S_n the group of all permutations on $\{1, 2, ..., n\}$. The members of S_n can be represented by strings of integers. For example when $n = 4$, $\sigma = 4132$ is a permutation, where $\sigma(1) = 4$, $\sigma(2) = 1$, $\sigma(3) = 3$, and $\sigma(4) = 2$.

We denote by $index(\sigma, i)$ the position of i in permutation σ (for example $index(\sigma, 1) = 2$).

With the elements of S_n, we define a directed graph where the nodes are the elements of S_n. In this digraph, there exists an edge between two nodes σ and σ' if and only if $\sigma = \alpha ij\beta$ and $\sigma' = \alpha ji\beta$, with α and β two partial orders, i, $j \in \{1, 2, ..., n\}$ with $index(\sigma, j) = index(\sigma, i) + 1$, and $i < j$. In other words, there is an edge between σ and σ' if these permutations are the same, except that there exist i and j, two consecutive jobs in σ with $i < j$, that are in the reverse order in σ'.

We call σ a *predecessor* of σ' (or σ' a *successor* of σ).

This digraph is a lattice, called the ***lattice of permutations*** or *permutohedron*. Fig 1.9 illustrates the lattices of permutations for $n = 3$ and $n = 4$.

To each permutation σ in the lattice can be associated a **level**. There are at most $\frac{n(n-1)}{2} + 1$ levels, from minimum value 0 (of permutation $(n, n-1, n-2, ..., 1)$) to maximum value $\frac{n(n-1)}{2}$ (of permutation $(1, 2, ..., n)$). We denote by $\kappa(\sigma)$ the level of permutation σ.

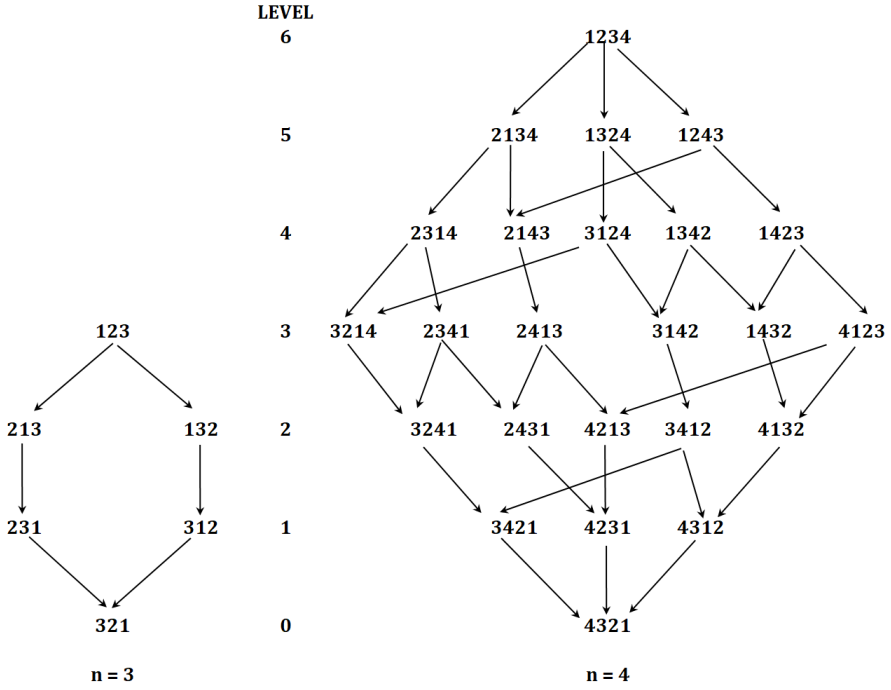

Figure 1.9: Lattice of permutations for $n = 3$ and $n = 4$

For a given permutation σ, we denote by $\Gamma(\sigma)$ the set of couples defined as follows:

$$\Gamma(\sigma) = \left\{(i,j) \in \{1,2,...,n\}^2 / i < j \text{ and } index(\sigma,i) < index(\sigma,j)\right\}$$

For example, in permutation $\sigma = 4132$, we have $\Gamma(\sigma) = \{(1,3),(1,2)\}$.

Properties We now report some properties associated with the lattice of permutations.

Property 1 *For any permutation σ, the level of σ is exactly its number of inversions from permutation at level 0, i.e., the number of times we have $i < j$ and $index(\sigma,i) < index(\sigma,j)$*

$$\kappa(\sigma) = |\Gamma(\sigma)|$$

Property 2 *Let consider a permutation σ. Any predecessor π of σ in the digraph is such that:*

$$\Gamma(\sigma) \subset \Gamma(\pi)$$

If we consider the elements of $\Gamma(\sigma)$ as a set of constraints associated to σ, we can say that all the predecessors of σ have to satisfy at least the same constraints as π. We then claim that $\Gamma(\sigma)$ gives the characteristics of all the predecessors of permutation σ in the digraph.

Relations between the lattice and the Permutohedron There are many definitions of a *permutohedron*. In mathematics, the permutohedron (also called permutahedron) of order n is an $(n-1)$-dimensional polytope embedded in an n-dimensional space, the vertices of which are formed by *permuting* the coordinates of the vector $(1, 2, 3, ..., n)$.

The relations between the *lattice of permutations* and *Permutohedron* are interesting for our problem.

Permutohedra were first studied by [Schoute, 1911]. The name "Permutohedron" was coined by [Osenstiehl, 1963]. Permutohedra are sometimes also called "permutation polytopes (or permutation polyhedron)" for a related polytope, for any polytope whose vertices are in 1-1 correspondence with the *permutations* of some sets [Bowman, 1972].

A permutohedron of order n, whose vertices are in one-to-one correspondence with the $n!$ permutations of the n jobs $J_1, J_2, ..., J_n$ in the lattice. Thus, each vertex of permutohedron represents a sequence in which the n jobs can be processed.

Fig. 1.10 is an example which illustrates *the equivalences* between the lattice and the permutohedron for $n = 4$.

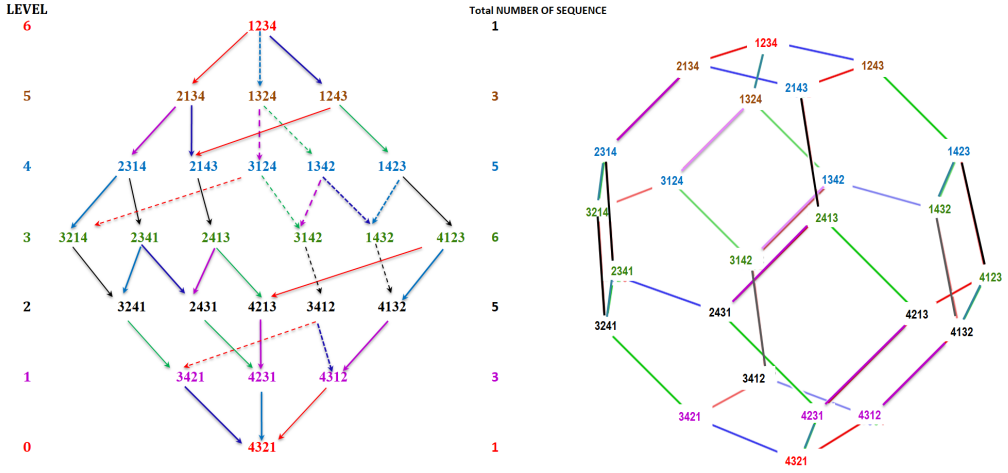

Figure 1.10: The lattice of permutations and the permutohedron for $n = 4$

The details of the equivalences between the lattice of permutations and the Permutohedron are the following:

- the symmetric group S_n acts on them by permutation of $\{1, 2, ..., n\}$,
- they have the same number of vertices, that is $n!$,
- each of the vertice is adjacent to $n-1$ others,
- the total number of edges is $(n-1)n!/2$,
- each edge connects two vertices that differ by swapping two adjacent elements,
- there are $n(n-1)/2$ linear independence vectors to contribute a permutohedron,

- the maximum level in the lattice is $n(n-1)/2$,
- there are a generating set of S_n with $n(n-1)/2$ generators which has the form $(ij), i \neq j$.

Thus, the total number of predecessors of a sequence with the root sequence 12...n in the lattice is exactly the total number of vertices from a vertex A to vertex 12...n (includes vertex 12...n) in the permutohedron.

However, the lattice has not much symmetric axis and **no** symmetric plane, while permutohedron has a lot symmetric axis and symmetric planes.

Besides, it is well known that there are many ways to deduce the properties of the geometry in the plane from the geometry of the space and vice versa. Thus, from the equivalences between the lattice and the permutohedron, we may deduce more characteristics of the lattice from the permutohedron in the space.

1.2.2.2 Application to scheduling problems

The resolution of some scheduling problems is equivalent to finding a sequence of jobs. It is the case for the problems where a sequence of jobs gives a schedule without ambiguity by simply scheduling the jobs in the order of the sequences, as early as possible. A sequence, i.e. a schedule, is completely characterized by a permutation of jobs. Therefore, we associate to each permutation of the lattice the corresponding permutation of jobs in the considered scheduling problem.

A lot of scheduling problems can be solved in polynomial time by the application of a simple rule, often called "priority rule". The application of this rule allows to know if in a sequence (permutation) π of jobs, it is better to schedule a job J_i before a job J_j or not. This is the case for some scheduling problems such as :

- the $1||L_{\max}$ that can be solved to opimality by sorting the jobs in the non decreasing order of their due dates (EDD rule) [Jackson, 1955] (see Section 1.1.5.2)
- the $1||\sum C_j$ and $1||\sum w_j C_j$ problems (see Section 1.1.5.1)
- the $1|r_j|C_{\max}$ problem that can be solved to opimality by sorting the jobs in the non decreasing order of their release dates (ERD rule)
- the $F2||C_{\max}$ problem that can be solved by the famous Johnson's rule [Johnson, 1954]
- etc.

As explained before, it is also well known that this scheduling problems generally have a lot of optimal solutions.

Let us assume that jobs are renumbered so that sequence $\sigma^\top = (J_1, J_2, ..., J_n)$ corresponds to the sequence returned by applying the sequencing rule. We will denote in the following by $\sigma^\perp$ the reverse sequence.

The idea is to find an optimal sequence, as far as possible from the root node $\sigma^\top$ of the lattice of permutations. For some evident pairwise echange arguments, it is clear that

all the predecessors of this sequence will be optimal sequences as well. And it is easy to give the characteristics of these sequences.

Definition 6 *We say that a sequence σ is **minimal** iff it is an optimal sequence (respecting the deadlines in our case) and none of its children are, i.e., any further swap of two consecutive jobs leads to a sub-optimal solution.*

Definition 7 *A **minimum sequence** is a minimal sequence at a minimum level in the lattice.*

It is assumed that a minimum sequence "covers" many optimal solutions.

Example 5 *Let consider a 4-job instance of a scheduling problem, where sequences (J_3, J_1, J_4, J_2) and (J_2, J_4, J_3, J_1) are the only minimal sequences (see Fig. 1.11)*

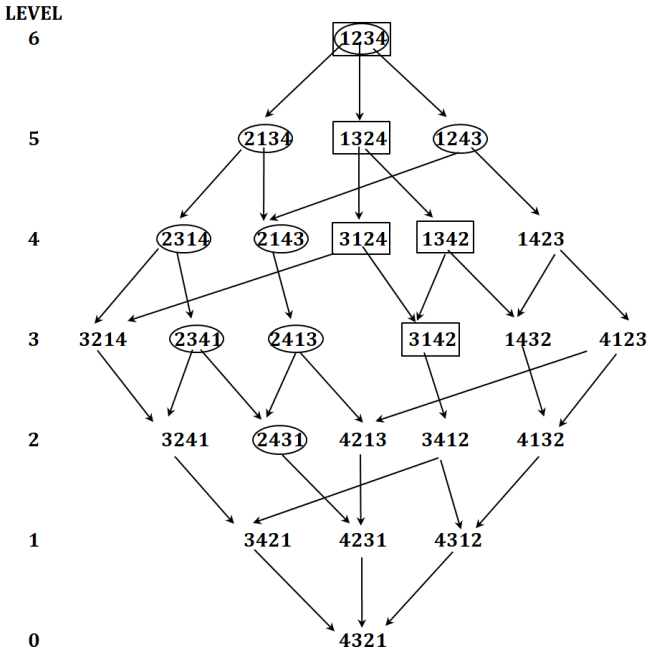

Figure 1.11: Minimal sequences in the lattice

Then, we know that the set of optimal solutions is exactly the set composed of these two sequences plus their predecessors in the lattice: (J_1, J_2, J_3, J_4), (J_2, J_1, J_3, J_4), (J_2, J_3, J_1, J_4), (J_2, J_3, J_4, J_1), (J_2, J_4, J_3, J_1), (J_2, J_1, J_4, J_3), (J_2, J_4, J_1, J_3), (J_1, J_2, J_4, J_3), (J_1, J_3, J_2, J_4), (J_3, J_1, J_2, J_4), (J_1, J_3, J_4, J_2), (J_3, J_1, J_4, J_2).

Notice that sequence (J_2, J_4, J_3, J_1) covers 8 optimal sequences (marked in circles), whereas (J_3, J_1, J_4, J_2) covers only 5 optimal sequences (marked in boxes).

The predecessors of $\sigma = (J_3, J_1, J_4, J_2)$ *are exactly all sequences such that:*

$$J_1 \prec J_4 \ (J_1 < J_4 \text{ and } index(\sigma, J_1) < index(\sigma, J_4)$$
$$J_1 \prec J_2 \ (J_1 < J_2 \text{ and } index(\sigma, J_1) < index(\sigma, J_2)$$
$$J_3 \prec J_4 \ (J_3 < J_4 \text{ and } index(\sigma, J_3) < index(\sigma, J_4)$$

We say that the sequences characterized by (J_3, J_1, J_4, J_2) *are those which respect:*

$$J_1 \prec J_4 \wedge J_1 \prec J_2 \wedge J_3 \prec J_4$$

Similarly, the sequences characterized by (J_2, J_4, J_3, J_1) *are such that:*

$$J_2 \prec J_4 \wedge J_2 \prec J_3$$

So, in this example, the sequences (J_3, J_1, J_4, J_2) *and* (J_2, J_4, J_3, J_1) *allow to characterize all optimal sequences, which are the sequences verifying*

$$(J_1 \prec J_4 \wedge J_1 \prec J_2 \wedge J_3 \prec J_4) \bigvee (J_2 \prec J_4 \wedge J_2 \prec J_3)$$

General method When one minimum sequence has been found, a set of sequences is characterized. It generally happens that this set of sequences is not the whole set of optimal sequences. Thus, in order to have the characteristics of all the optimal sequences, a new minimum sequence has to be found, in order to characterize other optimal sequences. The process for finding the characteristics of all the optimal sequences is iterative. At the beginning, the characteristic of the characterized sequences is empty. The two main steps are:

1. find a minimal sequence of jobs as deep as possible in the lattice (i.e. with minimum level) which is not already characterized,
2. update the characteristics of the solutions already characterized.

The process terminates when there is no more feasible solution.

1.2.2.3 Literature review about the lattice of permutation applied to the characterization of solutions for a scheduling problem

In many scheduling problems, a set of dominant solutions can be mapped to the set of permutations of jobs and consequently to the vertices of a permutohedron. Several studies have been conducted in this direction and using the lattice permutations as support to provide these features proved very interesting.

Some papers have already been published on this idea of characterization of solutions.

- the two-machine flowshop with makespan minimization is considered for the first time in [Benoit et Billaut, 2000]. The authors list all the minimal sequences by using a tree-search algorithm. Some computational experiments show that for $n = 11$ jobs, more than 1000 minimal sequences are needed to characterize the whole set of optimal sequences.

- the $1||L_{\max}$ and the $F2||C_{\max}$ problems are considered in [Billaut *et al.*, 2011]. An MILP is proposed to characterize the whole set of minimum sequences.
- in [Billaut et Lopez, 2011], an extension of optimality to ρ-approximated solutions is presented for the $1||L_{\max}$ and the $F2||C_{\max}$ problems. The authors presented the basic ideas and MILP models for finding the minimum sequences.
- in [Billaut *et al.*, 2012], an efficient resolution based on constraint programming is proposed for the $F2||C_{\max}$ problem, for finding the whole set of minimal sequences. Again, it is possible to see that for $n = 12$ jobs, there are more than 7000 sequences required to characterize the whole set of optimal sequences (more than $7.4\ 10^6$).

1.3 Problems studied in this thesis

The problem is to characterize a set of optimal sequences without having to enumerate them, by using the properties of the lattice of permutation. **In this thesis, we consider the $1|\widetilde{d}_j|-$ problem, assuming that the jobs are numbered in EDD order, and that at least one feasible sequence exists.** We focus on the problem of finding *a minimum sequence*, i.e. a minimal sequence as deep as possible in the lattice.

1.3.1 Introduction of new objective functions

We will show in Chapter 2 that the expression of the level in the lattice presents similarities with a new type of objective function, involving the (weighted) position of a job in the sequence.

Let us define

$$\Gamma_j(\sigma) = \{J_i / i > j \text{ and } index(\sigma, i) > index(\sigma, j)\}$$

i.e. $\Gamma_j(\sigma)$ is the set of jobs after J_j with an index greater than j. We introduce

$$N_j = |\Gamma_j(\sigma)|$$

the number of such jobs in σ. Of course, we have

$$\Gamma(\sigma) = \bigcup_{j=1}^{n} \Gamma_j(\sigma)$$

The level of σ in the lattice is exactly $\kappa(\sigma) = \sum_{j=1}^{n} N_j$. This is the notation that we consider to indicate the level of σ.

Therefore, we consider in this thesis the problem denoted by:

$$1|\widetilde{d}_j| \sum N_j \tag{1.3}$$

To be more precise, the notation should be $1|Feas.\widetilde{d}_j, \widetilde{d}_1 \leq \widetilde{d}_2 \leq ... \leq \widetilde{d_n}| \sum N_j$, to indicate that at least one solution is feasible, and that jobs are numbered in EDD order, but we prefer the notation (1.3) for more simplicity.

We use the following notations:

$$\sigma^{\top} = (J_1, J_2, ..., J_n) \tag{1.4}$$

$$\sigma^{\perp} = (J_n, J_{n-1}, ..., J_1) \tag{1.5}$$

We will denote by P_j the position of job J_j in the sequence. We will consider new objective functions such as $\sum P_j$, $\sum w_j P_j$ and the particular case where $w_j = j$.

Notice that all these objective functions are very specific since they do not depend on the jobs completion times, which is really unusual in the scheduling literature.

1.3.2 Outline of the thesis

In Chapter 2, we present in details the problem of finding a minimal sequence at minimum level. We present basic theory, mathematical expressions and properties. We consider also some particular cases.

In Chapter 3, we propose resolution methods for problem (1.3): non-polynomial time methods and polynomial time methods. Computational experiments are presented to see the performances of the algorithms.

The Chapter 4 deals with the minimization of the total weighted positions. We propose a complexity results and resolution methods.

Finally, the thesis terminates by a conclusion and some future research directions.

Chapter 2

A new sequencing problem: finding a minimum sequence

In this chapter, we investigate the main problem which is to find a minimal sequence at the minimum level.

2.1 Presentation of the level $\sum N_j$

The lattice structure has already been presented in Chapter 1. The level of a sequence (a permutation) has some properties, that make it possible to establish a link with other notions of the literature. More precisely, there is a direct connection between the level of a permutation in the lattice and the *Kendall's-τ distance*, and also with the *crossing number*.

These connections may lead to some interesting properties and results for our problem.

2.1.1 Relation with Kendall's-τ distance

The Kendall's-τ distance was created by Maurice Kendall [Kendall, 1938]. It is a metric that counts the number of pairwise disagreements between two ranking lists. Kendall's-τ distance is also called bubble-sort distance since it is equivalent to the number of swaps that the bubble sort algorithm would make to place one list in the same order as the other list.

Definition 8 *The Kendall's Tau ranking distance between two permutations π_1 and π_2 is*

$$K(\pi_1, \pi_2) = |\{(i,j) : i < j, (i \prec_{\pi_1} j \wedge j \prec_{\pi_2} i) \vee (j \prec_{\pi_1} i \wedge i \prec_{\pi_2} j)\}|$$

where $i \prec_\pi j$ means that i precedes j in π.

Notice that $K(\pi_1, \pi_2) = 0$ if the two lists are identical and $K(\pi_1, \pi_2) = n(n-1)/2$ if one list is the reverse of the other (with n the list size).

With the definition of the Level in Section 1.2.2.1, we have the relations between $\sum N_j$ and Kendall's-τ distance: $\sum N_j$ of a sequence π is exactly the Kendall's-τ distance between sequence π and the reverse EDD sequence $\sigma^{\perp} = (J_n, J_{n-1}, ..., J_2, J_1)$.

That is the reason why we can call the level $\sum N_j(\sigma)$, the Kendall's-τ distance of σ.

A simple algorithm based on *merge sort* requires $O(n \log n)$ computation time for computing the Kendall'sτ distance. A more advanced algorithm requires $O(n\sqrt{\log n})$ time [Chan, 2010].

Example 6 *Consider the sequence of 10 jobs* $\pi_1 = (J_{10}, J_8, J_7, J_5, J_1, J_9, J_6, J_3, J_2, J_4)$.

Then

$$\begin{aligned} K(\pi_1, \sigma^{\perp}) =& |\{(i,j) : i < j, (i \prec_{\pi_1} j \wedge j \prec_{\sigma^{\perp}} i) \vee (j \prec_{\pi_1} i \wedge i \prec_{\sigma^{\perp}} j)\}| \\ =& |\{(J_8, J_9), (J_7, J_9), (J_5, J_9), (J_5, J_6), (J_3, J_4), (J_2, J_4), (J_1, J_9), (J_1, J_6), (J_1, J_4), \\ & (J_1, J_3), (J_1, J_2)\}| \\ =& 11 \end{aligned}$$

The Level of sequence π_1 *is* $\sum N_j(\pi_1) = 11$.

2.1.2 Relation with the Crossing Number

Let us introduce the rank aggregation problem, also called the *crossings of permutations problem*. The problem is to combine several rankings π_1, ... π_k in order to find the best possible ranking π^*, where the notion of 'best ranking' lies in the objective function definition (a cost function involving the penalties between the π_i and π^*). The Kendall's-τ distance is commonly used and this distance has a correspondence in graph drawing and more particularly with the crossing minimization problem of two-layered graphs [Biedl *et al.*, 2009].

The following example highlights this correspondence.

Example 7 *For example, if we consider the same sequence as in the previous example, the equivalence with the crossing number is illustrated in Fig.2.1.*

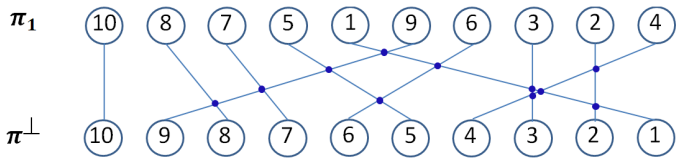

Figure 2.1: Crossing and Permutation

The level is 11, exactly equal to the total number of edge crossings.

Therefore, we can see that the level value $\sum N_j$ of a sequence π_1 is exactly the edge crossings number of two-layered graphs π_1 and $\sigma^{\perp}$.

The crossing number is very interesting and its applications appear in many aspects. It is a well known problem among graph theorists. The crossing minimization problem

in a planar surface is known to be NP-hard for general graphs [Garey et Johnson, 1983]. It is one of the most important studied problems in graph theory [Tollis *et al.*, 1999], [Hartmanis et Leeuwen, 2001], [Bennett, 2012], etc.. Besides, there are some known results in heuristics and meta-heuristics [Marti et Lanuna, 2003] and many applications in "On Metro-Line Crossing Minimization" [Argyriou *et al.*, 2010], etc.

Crossing number is also used to solve the *Kemeny optimal aggregation problem*, also known as the rank aggregation problem, which consists in finding a consensus ranking on a set of alternatives, based on the preferences of individual voters [Biedl *et al.*, 2006], [Biedl *et al.*, 2009].

There are some interesting results with the crossing number.

Crossing minimization and max crossing minimization problems Let consider a set of k permutations $P = \{\pi_1, ..., \pi_k\}$ and π^* is a "common" permutation. We denote by:

- $\kappa(P, \pi^*) = \sum_{i=1}^{k} K(\pi_i, \pi^*)$ the sum of Kendall's-τ distances from each permutation π_i to π^*.
- $\kappa_{max}(P, \pi^*) = max\{K(\pi_i, \pi^*) | \pi_i \in P\}$ the maximum of Kendall's-τ distances from each permutation π_i to π^*.

The problem of finding a permutation π^* minimizing $\kappa(P, \pi^*)$ is called the "crossing minimization problem", denoted by PCM-k problem. The problem of finding a permutation π^* minimizing $\kappa_{max}(P, \pi^*)$ is called the "max crossing minimization problem", denoted by PCM_{max}-k problem, with k the cardinality of set P. Permutation π^* corresponds to the best possible ranking, if the permutations of P correspond to some rankings.

Example 8 *For example, consider set* $P = \{\pi_1, \pi_2, \pi_3\}$, *with* $\pi_1 = (2314), \pi_2 = (1243)$, *and* $\pi_3 = (1324)$. *Consider a sequence* $\pi^* = (2143)$. *We have* $K(\pi_1, \pi^*) = 2$, $K(\pi_2, \pi^*) = 1$, $K(\pi_3, \pi^*) = 3$. *(We can see the results of Kendall's-*τ *distance in Fig. 2.2 and Fig 2.4)*

Then $\kappa(P, \pi^*) = 2 + 1 + 3 = 6$ *and* $\kappa_{max}(P, \pi^*) = K(\pi_3, \pi^*) = 3$

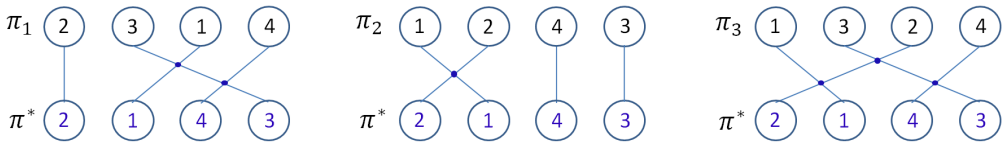

Figure 2.2: Example for $\pi^* = (2143)$

With $\pi^* = \pi^\top = (1234)$, *the results are illustrated in Fig. 2.3 and Fig 2.4. We have* $K(\pi_1, \pi^*) = 2$, $K(\pi_2, \pi^*) = 1$, $K(\pi_3, \pi^*) = 1$. *Then* $\kappa(P, \pi^*) = 2 + 1 + 1 = 4$ *and* $\kappa_{max}(P, \pi^*) = K(\pi_1, \pi^*) = 2$.

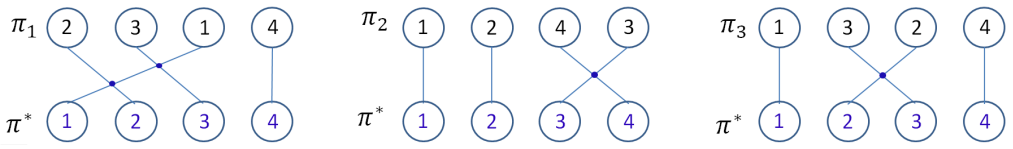

Figure 2.3: Example for $\pi^* = (1234)$

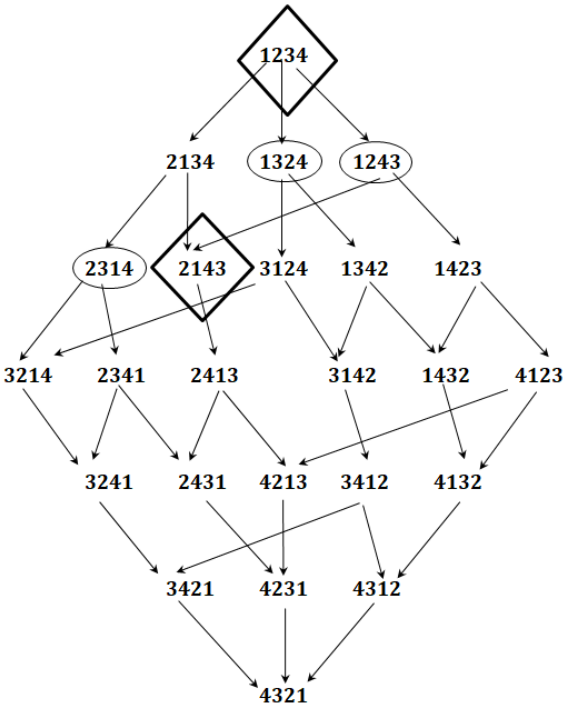

Figure 2.4: Kendall's-τ distance in the lattice

Complexity results :

- For $k \geq 4$ and k even, the *crossing minimization* problem PCM-k is NP-hard.
- For $k \geq 4$, the *max crossing minimization* problem PCM$_{max}$-k is NP-hard.
- PCM-3 and PCM$_{max}$-3 are still open problems.
- PCM-2 can be solved in $O(1)$ time, and PCM$_{max}$-2 takes at most $O(n^2 \log n)$ time.
- PCM-1= PCM$_{max}$-1 =0.

There is a $(2 - 2/k)$-approximation algorithm for problem PCM-k. There is a 2-approximation algorithm for problem PCM$_{max}$-k.

2.1.3 Relation with the One Sided Crossing Minimization problem

We consider a bipartite graph. The one sided crossing minimization (OSCM) problem consists of placing the vertices of one part on given positions on a straight line, and finding the positions of the vertices of the second part on a parallel line, then drawing the edges as straight lines. The objective is to minimize the number of pairwise edge crossings [Munoz *et al.*, 2002].

Munoz defines the OSCM problem for graphs with maximum degree k on the free side as the OSCM-k problem. They consider bipartite graphs being forests of stars ($k+1$ vertices and k leaves).

Example 9 *Let consider for example the input for the OSCM-4 problem presented in Fig. 2.5. In this bipartite graph, vertex u_1 has edges with vertices $\{v_1, v_4, v_6, v_{10}\}$, vertex u_2 has edges with vertices $\{v_3, v_7, v_9, v_{12}\}$ and vertex u_3 has edges with vertices $\{v_2, v_5, v_8, v_{11}\}$. Vertices v_i are fixed in a straight line, the problem is to find in which order putting vertices u_i so that the number of crossings is minimized.*

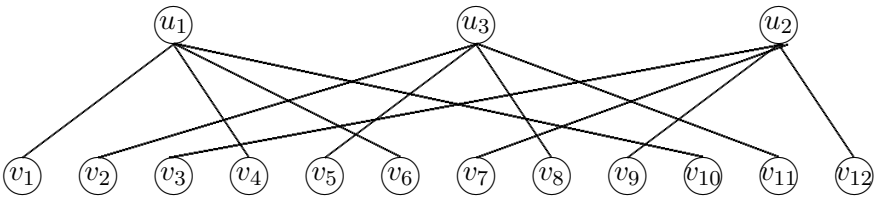

Figure 2.5: Example of input for the OSCM-4 problem

The sequence of vertices (u_1, u_3, u_2) has a number of crossings equal to 17.

The problem OSCM-2 can be solved in linear time. The problem OSCM-4 is proved NP-complete. The problem OSCM-3 is open.

Suppose that we are able to impose that some jobs have to be executed in a single block, then the problem $1|\widetilde{d_j}|\sum N_j$ would be NP-hard. But to impose such batches, we need to introduce release times, but in this case, there is no need of the OSCM-4 problem to say that problem $1|r_j, \widetilde{d_j}|\sum N_j$ is NP-hard.

2.1.4 Relation with the Checkpoint Ordering Problem

The Checkpoint Ordering Problem is introduced by P. Hungerlander [Hungerlaner, 2017]. We consider a set of n departments, where a length l_i and a weight w_i is associated to each department i ($1 \leq i \leq n$). A checkpoint is assigned to a fixed position (left-aligned or at center for example). For a given permutation π or sequence of departments, the distance between the center of department i and the checkpoint is denoted by $z_i(\pi)$. The problem is to find a permutation π^* of the departments so that the weighted sum $\sum_{i=1}^{n} w_i z_i(\pi^*)$ is minimum for all the possible permutations.

Example 10 *Let consider an instance with $n = 4$ departments with the lenghts $l = (4, 6, 3, 5)$ and the weights $w = (2, 4, 1, 3)$. Let consider the permutation (2,1,4,3) illustrated in Fig. 2.6. The checkpoint C is put at the midle (9). The center of department 2 is at a distance 6 to C ($z_2 = 6$), then $z_1 = 1$, etc.*

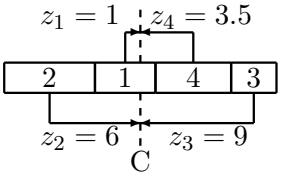

Figure 2.6: Example for the COP problem

The evaluation of this permutation is equal to $\sum w_i z_i = 45.5$.

This problem is proved NP-hard by reduction from PARTITION problem. This problem presents some connexions with the Linear Ordering Problem (LOP: find a tournament of maximum total edge weight without directed cycles of a complete weighted directed graph).

In this problem, departments can be considered as jobs and the checkpoint as a common deadline. However, the distance can be associated to a completion time (if the checkpoint is put at the beginning), not to the position.

The LOP is NP-complete, but the relation with our problem leads to a graph with specific weights (0 or 1) which is easy to solve (permutation $\sigma^{\perp}$ is optimal). The difficulty is to introduce the deadlines and the durations of jobs.

2.2 Mathematical expressions and properties

We give in this section some mathematical expressions of the objective function $\sum N_j$, and some first properties.

Let remember that we consider a set of n jobs $\{J_1, J_2, ..., J_n\}$ to schedule on a single machine. Preemption is not allowed and the machine can perform only one job at a time. To each job J_j is associated a processing time p_j and a deadline $\tilde{d}_j$. It is assumed that the sequence is feasible (maximum lateness equal to 0) and that the jobs are numbered in EDD order (notice that it can be done in $O(n \log n)$ time).

Let remember that $\sum N_j$ corresponds to the level of sequence σ in the lattice of permutations. Finding a sequence as deep as possible in this lattice, or finding a sequence as far as possible from sequence $\sigma^{\top} = (J_1, J_2, ..., J_n)$ is the same and is equivalent to minimize the objective function $\sum N_j$.

There are several ways to give a mathematical expression of $\sum N_j$.

2.2.1 Expression of N_j based on position variables

Let us define the following boolean variables (see Section 1.1.3.1):

$$x_{j,k} = \begin{cases} 1 & \text{if job } J_j \text{ is in position } k \\ 0 & \text{otherwise} \end{cases}$$

The expression of N_j is the following:

$$N_j = \sum_{k=1}^{n} \sum_{i=j+1}^{n} \sum_{h=k+1}^{n} x_{i,h} x_{j,k} \tag{2.1}$$

If J_j is in position k ($x_{j,k} = 1$) it is equal to the number of jobs after J_j – i.e. at a position h greater than k – with an index i greater than j.

Therefore, the expression of $\sum N_j$ is:

$$\sum N_j = \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{i=j+1}^{n} \sum_{h=k+1}^{n} x_{i,h} x_{j,k} \tag{2.2}$$

This expression is quadratic and in an MILP model, it is possible to linearize this expression [Billaut et Lopez, 2011] and [Billaut *et al.*, 2011].

Let us define:

$$y_{j,k} = \sum_{i=j+1}^{n} \sum_{h=k+1}^{n} x_{i,h}$$

$y_{j,k}$ is the contribution of J_j to the objective function, if J_j is in position k. We have:

$$\sum N_j = \sum_{j=1}^{n} \sum_{k=1}^{n} y_{j,k} x_{j,k} \tag{2.3}$$

Let us consider the function Z' defined by:

$$Z' = \sum_{j=1}^{n} \sum_{k=1}^{n} y_{j,k}$$

Notice that Z' is related, but not equivalent to the expression of $\sum N_j$.

Proposition 3 *Z' is a function which depends only on the positions of the jobs in the sequence.*

Proof. Two sequences or permutations play a particular role in the problem. Sequence $\sigma^{\top} = (1, 2, ..., n)$ at level $n(n-1)/2$ and sequence $\sigma^{\perp} = (n, n-1, ..., 1)$ at level 0. Suppose that $n = 5$. The matrices of variables $x_{j,k}$ associated to sequence $\sigma^{\top}$ and $\sigma^{\perp}$ are the following.

$$MX(\sigma^{\top}) = \begin{pmatrix} 1 & . & . & . & . \\ . & 1 & . & . & . \\ . & . & 1 & . & . \\ . & . & . & 1 & . \\ . & . & . & . & 1 \end{pmatrix} \quad MX(\sigma^{\perp}) = \begin{pmatrix} . & . & . & . & 1 \\ . & . & . & 1 & . \\ . & . & 1 & . & . \\ . & 1 & . & . & . \\ 1 & . & . & . & . \end{pmatrix}$$

Associated to $\sigma^\top$ and $\sigma^\perp$, we have the following MY matrices (matrices of variables $y_{j,k}$).

$$MY(\sigma^\top) = \begin{pmatrix} 4 & 3 & 2 & 1 & . \\ 3 & 3 & 2 & 1 & . \\ 2 & 2 & 2 & 1 & . \\ 1 & 1 & 1 & 1 & . \\ . & . & . & . & . \end{pmatrix} \quad MY(\sigma^\perp) = \begin{pmatrix} 3 & 2 & 1 & . & . \\ 2 & 1 & . & . & . \\ 1 & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \end{pmatrix}$$

We can see that the level of $\sigma^\top$ is 10 (sum of the diagonal terms in $MY(\sigma^\top)$) and the level of $\sigma^\perp$ is 0 (sum of the terms in the opposite diagonal).

Let focus on $MY(\sigma^\top)$. To find Z', we have to compute the sum of all the elements of this matrix. We can rewrite this sum as the following sum:

$$MY(\sigma^\top) = \begin{pmatrix} 4 & 3 & 2 & 1 & . \\ 4 & 3 & 2 & 1 & . \\ 4 & 3 & 2 & 1 & . \\ 4 & 3 & 2 & 1 & . \\ 4 & 3 & 2 & 1 & . \end{pmatrix} + \begin{pmatrix} . & . & . & . & . \\ -1 & . & . & . & . \\ -2 & -1 & . & . & . \\ -3 & -2 & -1 & . & . \\ -4 & -3 & -2 & -1 & . \end{pmatrix}$$

Therefore, we deduce that:

$$\begin{aligned} Z'(\sigma^\top) &= n \times \left(\frac{n(n-1)}{2}\right) - \left(\sum_{i=1}^{n-1}\sum_{j=1}^{i} j\right) \\ &= n \times \left(\frac{n(n-1)}{2}\right) - \left(\sum_{i=1}^{n-1} \frac{i(i+1)}{2}\right) \\ &= \frac{1}{2}n^2(n-1) - \frac{1}{2}\sum_{i=1}^{n-1} i - \frac{1}{2}\sum_{i=1}^{n-1} i^2 \\ &= \frac{1}{2}n^2(n-1) - \frac{1}{2}.\frac{1}{2}(n-1)n - \frac{1}{2}.\frac{1}{6}n(n-1)(2n-1) \\ &= \frac{1}{2}n(n-1)\left[n - \frac{1}{2} - \frac{1}{6}(2n-1)\right] \\ &= \frac{1}{2}n(n-1)\frac{1}{6}\left[6n - 3 - 2n + 1\right] \\ &= \frac{1}{6}n(n-1)(2n-1) \end{aligned}$$

Similarly, we can prove that:

$$Z'(\sigma^\perp) = \frac{1}{6}n(n-1)(n-2)$$

Each time a variable $x_{j,k}$ is set to 1, it generates a modification in matrix MY that can be easily analysed (see Fig. 2.7). If $k > j$, there is '+1' in the submatrix of MY between columns j and $k-1$ and lines 1 to $j-1$. So the value of Z' becomes $Z'(\sigma^\top)+(j-1)(k-j)$. Similarly, if $k < j$, there is '−1' in the submatrix of MY between columns k and $j-1$ and lines 1 to $j-1$. So the value of Z' becomes $Z'(\sigma^\top) - (j-k)(j-1)$.

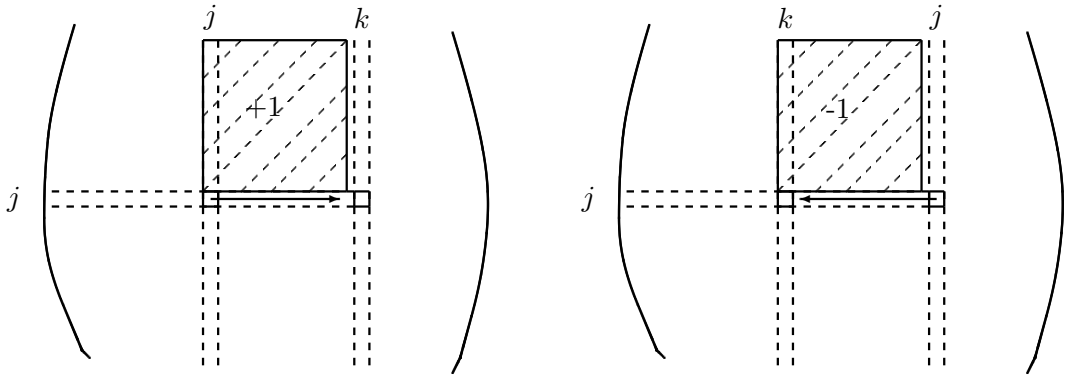

Figure 2.7: Modification of matrix MY

$$
\begin{aligned}
Z' &= Z'(\sigma^\top) + \sum_{j=1}^{n} \Bigg(\sum_{k=j+1}^{n} (k-j)(j-1)x_{j,k} - \sum_{k=1}^{j-1} (j-k)(j-1)x_{j,k} \Bigg) \\
&= Z'(\sigma^\top) + \sum_{j=1}^{n}\sum_{k=1}^{n} (k-j)(j-1)x_{j,k} \\
&= Z'(\sigma^\top) + \sum_{j=1}^{n}\sum_{k=1}^{n} kjx_{j,k} - \sum_{j=1}^{n}\sum_{k=1}^{n} kx_{j,k} - \sum_{j=1}^{n}\sum_{k=1}^{n} j^2x_{j,k} + \sum_{j=1}^{n}\sum_{k=1}^{n} jx_{j,k}
\end{aligned}
$$

$$
\Leftrightarrow Z' = Z'(\sigma^\top) + \sum_{j=1}^{n}\sum_{k=1}^{n} kjx_{j,k} - \sum_{k=1}^{n} k(\sum_{j=1}^{n} x_{j,k}) - \sum_{j=1}^{n} j^2(\sum_{k=1}^{n} x_{j,k}) + \sum_{j=1}^{n} j(\sum_{k=1}^{n} x_{j,k})
$$

Because $\sum_{j=1}^{n} x_{j,k} = 1$ and $\sum_{k=1}^{n} x_{j,k} = 1$, we have:

$$
\begin{aligned}
Z' &= Z'(\sigma^\top) + \sum_{j=1}^{n}\sum_{k=1}^{n} kjx_{j,k} - \sum_{k=1}^{n} k - \sum_{j=1}^{n} j^2 + \sum_{j=1}^{n} j \\
&= Z'(\sigma^\top) + \sum_{j=1}^{n}\sum_{k=1}^{n} kjx_{j,k} - \frac{1}{2}n(n+1) - \frac{1}{6}n(n+1)(2n+1) + \frac{1}{2}n(n+1)
\end{aligned}
$$

$$\begin{aligned}
\Leftrightarrow Z' &= \frac{1}{6}n(n-1)(2n-1) + \sum_{j=1}^{n}\sum_{k=1}^{n} kjx_{j,k} - \frac{1}{6}n(n+1)(2n+1) \\
&= \frac{1}{6}n\Big[(n-1)(2n-1) - (n+1)(2n+1)\Big] + \sum_{j=1}^{n}\sum_{k=1}^{n} kjx_{j,k} \\
&= \frac{1}{6}n(2n^2 - n - 2n + 1 - 2n^2 - n - 2n - 1) + \sum_{j=1}^{n}\sum_{k=1}^{n} kjx_{j,k} \\
&= -n^2 + \sum_{j=1}^{n}\sum_{k=1}^{n} jkx_{j,k}
\end{aligned}$$

Notice that this expression is equivalent to:

$$Z' = \sum_{j=1}^{n}\sum_{k=1}^{n}(j-1)(k-1)x_{j,k}$$

Indeed:

$$\begin{aligned}
Z' &= \sum_{j=1}^{n}\sum_{k=1}^{n}(j-1)(k-1)x_{j,k} \\
&= \sum_{j=1}^{n}\sum_{k=1}^{n} jkx_{j,k} - \sum_{j=1}^{n}\sum_{k=1}^{n} kx_{j,k} - \sum_{j=1}^{n}\sum_{k=1}^{n} jx_{j,k} + \sum_{j=1}^{n}\sum_{k=1}^{n} x_{j,k} \\
&= \sum_{j=1}^{n}\sum_{k=1}^{n} jkx_{j,k} - \sum_{k=1}^{n} k\sum_{j=1}^{n} x_{j,k} - \sum_{j=1}^{n} j\sum_{k=1}^{n} x_{j,k} + n \\
&= \sum_{j=1}^{n}\sum_{k=1}^{n} jkx_{j,k} - \sum_{k=1}^{n} k - \sum_{j=1}^{n} j + n \\
&= \sum_{j=1}^{n}\sum_{k=1}^{n} jkx_{j,k} - n(n+1)/2 - n(n+1)/2 + n \\
&= \sum_{j=1}^{n}\sum_{k=1}^{n} jkx_{j,k} - n(n+1) + n
\end{aligned}$$

$$\Leftrightarrow Z' = \sum_{j=1}^{n}\sum_{k=1}^{n} jkx_{j,k} - n^2$$

In this expression, $\sum_{k=1}^{n} kx_{j,k}$ is exactly the position of job J_j in the sequence.
So, if we define by P_j the position of job J_j, then:

$$\text{minimizing } Z' \text{ is equivalent to minimize } \sum_{j=1}^{n} jP_j$$

So $\sum_{j=1}^{n} y_{j,k}$ is a function related to the positions of the jobs in the sequence.

It means in some sense, that the minimization of $\sum N_j$ is also related to the minimization of $\sum_{j=1}^{n} jP_j$.

Remember that minimizing Z' is not equivalent to minimize $\sum N_j$. This is illustrated in the following example.

Example 11 *Let consider the sequences of the lattice for $n = 5$, given in Table 2.1. For each sequence, it is possible to compute the level corresponding to the level $\sum N_j$ and the value of Z'.*

We can see for example that sequence $(5, 2, 1, 3, 4)$ has a value of $\sum N_j$ equal to 5 and a value of Z' equal to 19, whereas sequence $(1, 5, 4, 3, 2)$ has a value of $\sum N_j$ equal to 4 but a value of Z' equal to 20. Even if $\sum N_j$ and Z' seem to be correlated, it is not possible to say that $\sum N_j(\sigma_1) < \sum N_j(\sigma_2) \Leftrightarrow Z'(\sigma_1) < Z'(\sigma_2)$.

We notice that such an example cannot be found for $n = 4$.

2.2.2 Expression of N_j based on precedence variables

Let us define now consider **precedence variables** (see Section 1.1.3.1).

$z_{i,j} = \begin{cases} 0 & \text{if job } J_i \text{ precedes } J_j \\ 1 & \text{otherwise} \end{cases}$

We have some trivial constraints:

$$z_{j,j} = 1, \forall j$$
$$z_{j,i} + z_{i,j} = 1, \forall i \neq j$$

With these variables, the expression of N_j and the level $\sum N_j$ are the following:

$$N_j = \sum_{i=j+1}^{n} z_{i,j} \tag{2.4}$$

$$\sum N_j = \sum_{j=1}^{n} \sum_{i=j+1}^{n} z_{i,j} \tag{2.5}$$

The position of job J_j is given by:

$$P_j = \sum_{i=1}^{n} z_{j,i} \tag{2.6}$$

	$\sum N_j$	Z'		$\sum N_j$	Z'		$\sum N_j$	Z'
12345	10	30	15324	6	23	24513	5	19
12354	9	29	15342	5	21	24531	4	17
12435	9	29	15423	5	21	25134	6	22
12453	8	27	15432	4	20	25143	5	21
12534	8	27	21345	9	29	25314	5	20
12543	7	26	21354	8	28	25341	4	17
13245	9	29	21435	8	28	25413	4	18
13254	8	28	21453	7	26	25431	3	16
13425	8	27	21534	7	26	31245	8	27
13452	7	24	21543	6	25	31254	7	26
13524	7	25	23145	8	27	31425	7	25
13542	6	23	23154	7	26	31452	6	22
14235	8	27	23415	7	24	31524	6	23
14253	7	25	23451	6	20	31542	5	21
14325	7	26	23514	6	22	32145	7	26
14352	6	23	23541	5	19	32154	6	25
14523	6	22	24135	7	25	32415	6	23
14532	5	21	24153	6	23	32451	5	19
15234	7	24	24315	6	23	32514	5	21
15243	6	23	24351	5	19	32541	4	18

	$\sum N_j$	Z'		$\sum N_j$	Z'		$\sum N_j$	Z'
34125	6	22	42315	5	21	51423	4	17
34152	5	19	42351	4	17	51432	3	16
34215	5	21	42513	4	17	52134	5	19
34251	4	17	42531	3	15	52143	4	18
34512	4	15	43125	5	21	52314	4	17
34521	3	14	43152	4	18	52341	3	14
35124	5	19	43215	4	20	52413	3	15
35142	4	17	43251	3	16	52431	2	13
35214	4	18	43512	3	14	53124	4	17
35241	3	15	43521	2	13	53142	3	15
35412	3	14	45123	4	15	53214	3	16
35421	2	13	45132	3	14	53241	2	13
41235	7	24	45213	3	14	53412	2	12
41253	6	22	45231	2	12	53421	1	11
41325	6	23	45312	2	12	54123	3	14
41352	5	20	45321	1	11	54132	2	13
41523	5	19	51234	6	20	54213	2	13
41532	4	18	51243	5	19	54231	1	11
42135	6	23	51324	5	19	54312	1	11
42153	5	21	51342	4	17	54321	0	10

Table 2.1: Values of $\sum N_j$ and Z' for all the sequences of size 5

Therefore:

$$\begin{aligned} P_j &= \sum_{\substack{i=1 \\ i\neq j}}^{n}(1-z_{i,j}) + z_{j,j} = (n-1) - \sum_{\substack{i=1 \\ i\neq j}}^{n} z_{i,j} + 1 \\ &= n - \sum_{\substack{i=1 \\ i\neq j}}^{n} z_{i,j} = n - (\sum_{i=1}^{j-1} z_{i,j} + \sum_{i=j+1}^{n} z_{i,j}) \\ &= n - (\sum_{i=1}^{j-1} z_{i,j} + N_j) \end{aligned}$$

$$\Rightarrow N_j + P_j = n - \sum_{i=1}^{j-1} z_{i,j} \tag{2.7}$$

This expression gives a relation between the level and the position. Let us define $Q_j = \sum_{i=j+1}^{n} z_{j,i}$.

We have

$$\begin{aligned} N_j + Q_j &= \sum_{i=j+1}^{n} z_{i,j} + \sum_{i=j+1}^{n} z_{j,i} = \sum_{i=j+1}^{n} (z_{i,j} + z_{j,i}) \\ &= n - j \end{aligned}$$

$$\Rightarrow N_j + Q_j = n - j \tag{2.8}$$

We deduce also that

$$\begin{aligned} \sum_{j=1}^{n}(N_j + Q_j) &= \sum_{j=1}^{n}(n-j) \\ &= n(n-1)/2 = n(n+1)/2 - n \\ &= \sum_{j=1}^{n} P_j - n \end{aligned}$$

$$\Rightarrow \sum_{j=1}^{n}(N_j + Q_j) = \sum_{j=1}^{n} P_j - n \tag{2.9}$$

The example below illustrates the results presented above.

Example 12 *Let consider $n = 10$ jobs, and a sequence $\sigma = (J_8, J_{10}, J_5, J_1, J_9, J_6, J_2, J_7, J_4, J_3)$. The matrix MZ of $z_{i,j}$ is presented in Table 2.2.*

We have two ways to compute the position of a job: by index i or j. In the table, we present P_i by using simply the formulation $P_i = \sum_{j=1}^{n} y_{i,j}$ (formulation 2.6).

$MZ(\sigma)$											Q_j	P_i
1	1				1			1		1	3	4
2	*1*	**1**			1	1		1	1	1	5	7
3	*1*	*1*	**1**	1	1	1	1	1	1	1	7	10
4	*1*	*1*		**1**	1	1	1	1	1	1	6	9
5					**1**			1		1	2	3
6	*1*				*1*	**1**		1	1	1	3	6
7	*1*	*1*			*1*	*1*	**1**	1	1	1	3	8
8								**1**			0	1
9	*1*				*1*			*1*	**1**	1	1	5
10								*1*		**1**	0	2
$N_j = \sum_{i=j+1}^{n} y_{i,j}$	6	3	0	0	3	1	0	2	0	0	$\sum(N_j + Q_j) = 45$	$\sum P_j = 55$

Table 2.2: Relations between N_j and P_j

We can also obtain the position of a job by index j of formulation $P_j = n-(\sum_{i=1}^{j-1} y_{i,j} + N_j)$ (formulation 2.7), for example:

$j = 1, P_1 = 10 - N_1 - \sum_{i=1}^{0} y_{i,1} = 10 - 6 - y_{0,1} = 4$

$j = 2, P_2 = 10 - N_2 - \sum_{i=1}^{1} y_{i,2} = 10 - 3 - y_{1,2} = 7 - 0 = 7$

$j = 3, P_3 = 10 - N_3 - \sum_{i=1}^{2} y_{i,3} = 10 - 0 - y_{1,3} - y_{2,3} = 10$

$j = 4, P_4 = 10 - N_4 - \sum_{i=1}^{3} y_{i,4} = 10 - 0 - y_{1,4} - y_{2,4} - -y_{3,4} = 10 - 1 = 9$

$j = 5, P_5 = 10-N_5-\sum_{i=1}^{4} y_{i,5} = 10-3-y_{1,5}-y_{2,5}-y_{3,5}-y_{4,5} = 10-3-1-1-1-1 = 3$

etc,

Moreover, it is easy to check the other formulations on this example.

Thus, we can also easily deduce the positions of jobs in the sequence by using the precedence variables. Two MILP based on these variables are presented in the Section devoted to resolution methods.

2.2.3 Properties

We give in this section some properties related to the objective function $\sum N_j$.

2.2.3.1 General properties of the feasible sequence

We assume that the *root sequence* $\sigma^{\top}$ of the lattice was obtained by using EDD algorithm. We look for a minimum sequence (minimal sequence at minimum level) denoted σ^* in the lattice, since this solution is likely to cover many optimal solutions.

Obviously, a sequence σ is feasible for $1|\widetilde{d}_j|-$ problem if all the jobs respect their deadlines, i.e. $C_j \leq \widetilde{d}_j$, $\forall j \in \{1, ..., n\}$.

We call **critical job** a job that satisfies two conditions:

1. in $\sigma^{\top}$ it completes at its deadline ($C_j = \widetilde{d}_j$).
2. in a set of jobs with the same deadline, its index is the smallest.

Property 3 *In the root sequence $\sigma^{\top}$, consider critical job J_j, we have:*

a. *$\forall k < j$ (J_k precedes J_j) and $\widetilde{d}_k < \widetilde{d}_j$, J_k can not be scheduled after J_j in any feasible sequence.*

b. *$\forall l > j$ (J_l after J_j) and $\widetilde{d}_l > \widetilde{d}_j$, J_l can not be scheduled before J_j in any feasible sequence.*

Proof.

a. $\forall k < j$ (J_k precedes J_j) and $\widetilde{d}_k < \widetilde{d}_j$, if J_k is moved to be scheduled after J_j in a new sequence, we have $C'_k \geq \widetilde{d}_j$ (The equation occurs when J_k is scheduled just after J_j). But $\widetilde{d}_k < \widetilde{d}_j \Rightarrow C'_k \geq \widetilde{d}_j > \widetilde{d}_k$, which is impossible.

b. All the jobs J_k before J_j must be performed before J_j, and $C_j = \widetilde{d}_j$. So, there is no place for another job (J_l) before J_j.

Property 4 *In an EDD sequence, let consider two consecutive critical jobs J_i and J_j. We denote by $W[i+1, j] = \{i+1, i+2, ..., j\}$ the set of positions from $i+1$ to j. In any feasible solution, all jobs located on the positions that belong to $W[i+1, j]$ cannot move to other outside positions.*

Proof Apply Property 3 a. and b. to the jobs in $W[i+1, j]$, and we have the conclusion. Figure 2.8 illustrates for this property.

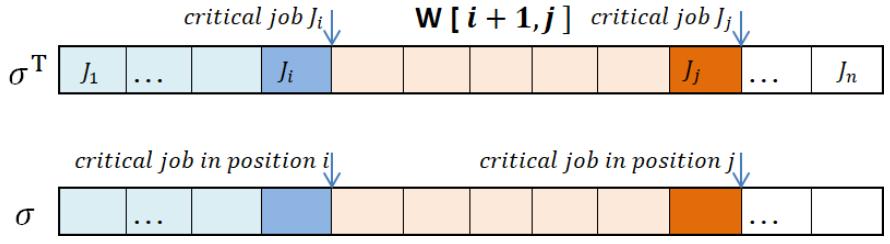

Figure 2.8: Property 4

From property 4, we can focus on the intervals which separate the critical jobs. We only need to consider the sequences with one critical job at the end of each sequence.

Property 5 *Suppose that the sequence $\sigma^{\top} = \sigma_1\sigma_2\sigma_3$, with σ_2 being a partial sequence corresponding to an interval $W[i+1, j]$, as presented in Property 4. The number of crossings between σ_1, σ_2 and σ_3 do not depend on the positions of the jobs in each interval.*

Proof. Call a, b, c the number of jobs in σ_1, σ_2 and σ_3, respectively (see Fig. 2.9). All the jobs in σ_1 will cross the jobs in σ_2 and in σ_3, generating $a(b+c)$ crossings. Similarly, all the jobs in σ_3 will cross the jobs in σ_2. Therefore, at least there are $ab + ac + bc$ crossings in any feasible solution. Therefore, the optimization has to focus on the minimization of crossings inside each interval.

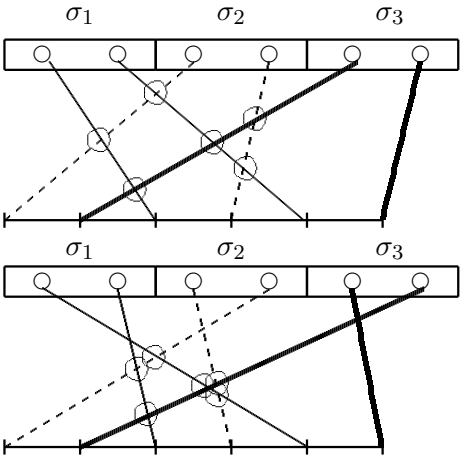

Figure 2.9: Illustration of Property 5

2.2.3.2 Batches

We introduce in this section the notion of batch.

Property 6 *A minimal solution can always be decomposed into a succession of batches, defined as follows:*

- *the "head" of the batch is the last job of the batch,*
- *all the jobs in the batch are in decreasing numbering order and have an index greater than the head.*

Therefore, the index of the heads are increasing, starting with index 1.

Proof. Let consider a minimal solution. The first batch $\mathcal{B}_1$ is composed by the first jobs up to job J_1. Suppose that these jobs are not in their reverse numbering order. Then, reordering these jobs in their reverse numbering order does not violate the deadlines and is better for the objective function. It would lead to a better solution, which is impossible. So these jobs are in their reverse numbering order. The next batch $\mathcal{B}_2$ starts after job J_1 and finishes with the job with the smallest index in $\{1, 2, ..., n\} \setminus \mathcal{B}_1$. The same pairwise interchange reasoning can be applied for this batch, and for the other batches. At the end, the proposition holds.

Example 13 *Consider the sequence* $(J_8, J_7, J_3, J_1, J_2, J_4, J_6, J_5, J_{10}, J_9)$*. This sequence is composed by 5 batches: batch* $\mathcal{B}_1$ *contains jobs* $\{J_8, J_7, J_3, J_1\}$*, batch* $\mathcal{B}_2$ *contains only job*

J_2, batch $\mathcal{B}_3$ contains only job J_4, batch $\mathcal{B}_4$ contains jobs J_6 and J_5 and finally, batch $\mathcal{B}_5$ contains jobs J_{10} and J_9.

$$(J_8, J_7, J_3, \boxed{J_1}, \boxed{J_2}, \boxed{J_4}, J_6, \boxed{J_5}, J_{10}, \boxed{J_9})$$

Corollary 1 *We have the following corollary:*

1. *If a solution with one batch is feasible, this solution is necessarily $\sigma^{\perp}$ at level 0.*
2. *A solution with n batches always exists, is unique, and is $\sigma^{\top}$ at level $\frac{n(n-1)}{2}$.*

A natural question comes: "Is a solution with $k+1$ batches necessarily at a level higher than a solution with k batches?" The answer is no. For example, let consider five jobs. Sequence $(J_1, J_3, J_2, J_5, J_4)$ is at level 8 and has 3 batches but sequence $(J_1, J_5, J_2, J_3, J_4)$ with 4 batches is only at level 7. So minimizing the level is not equivalent to minimize the number of batches.

This particular structure of a minimal sequence allows to reduce the search space. We illustrate in Fig. 2.10 the reduction of the search space for $n = 3$ and $n = 4$. But of course, the search space remains exponential.

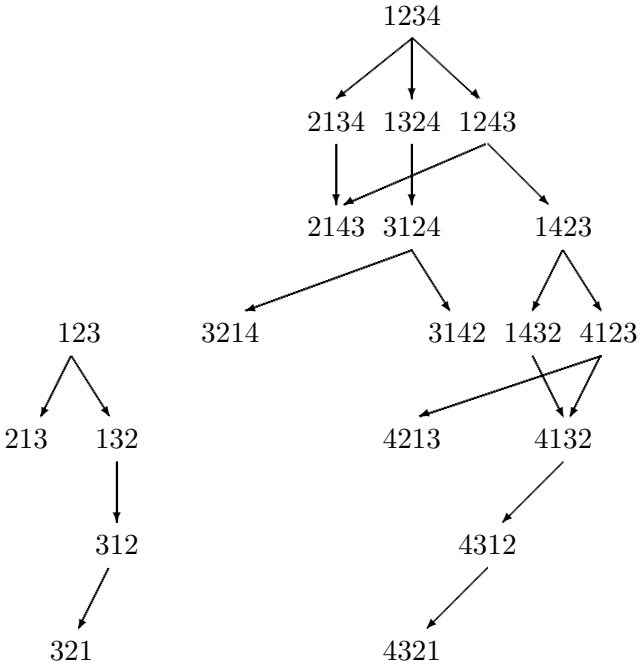

Figure 2.10: Lattice of permutation for $n = 3$ and $n = 4$ restricted to interesting sequences

We propose now to determine the minimum number of batches in a feasible solution.

We introduce the algorithm Min # B presented in Alg. 6. This algorithm allows to identify the minimum number of "head" jobs, i.e. the minimum number of batches.

Algorithm 6 Min # B

$\sigma^\top = (J_1, ..., J_n)$
Compute $C_j^{\sigma^\top}$, $\forall j \in \{1, 2, ..., n\}$
$k = 0$, $S = \emptyset$, $t = 1$
while $t \leq n$ **do**
  $i = t + 1$
  $bool = True$
  while $i \leq n$ and $bool$ **do**
    if $C_i^{\sigma^\top} > \widetilde{d}_t$ **then**
      $bool = False$
      $S = S \cup \{J_t\}$
      Put the jobs between J_t and J_{i-1} in the same batch
      $t = i$
    else
      $i = i + 1$
return $(|S|)$

Property 7 *Two "head" jobs returned by the algorithm cannot belong to the same batch. Therefore, the number of head jobs is the minimum number of batches in a feasible sequence.*

Proof. Let consider sequence $\sigma^\top$ and the sequence built by Alg. 2.11. We denote by t_j the first job in batch number j and by $t_{j+1} - 1$ the last job in this batch.

(a) $t_{j-1} + 1$ cannot be put before $\tilde{d}_{t_{j-2}+1}$, i.e. it cannot be put in batch $j - 1$ because it would be in first position in this batch and $t_{j-2} + 1$ would not respect its deadline. Similarly, $t_j + 1$ cannot be put before job $t_{j-1} + 1$.

(b) if $t_{j-1} + 1$ is put in the next batch $(j + 1)$, it would be put in the last position. In this case, we have $t_j + 1$ before $t_{j-1} + 1$, which is not possible.

(c) Therefore, at least $t_{j-1} + 1$ is alone in its batch, but the batch do exist. So we cannot reduce the number of batches.

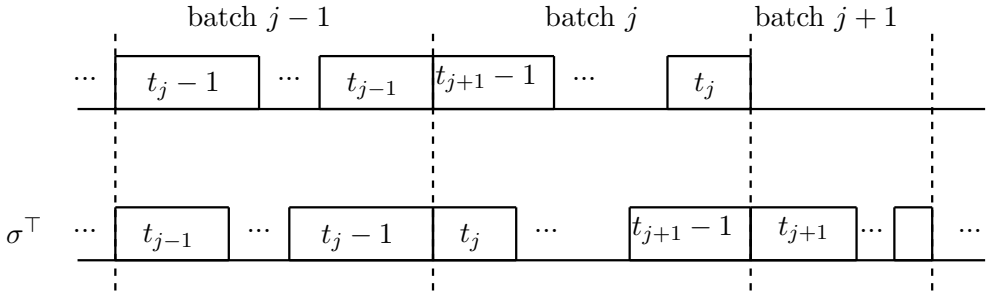

Figure 2.11: Illustration of the proof of Proposition 7

Example 14 *Let consider an instance with* $n = 10$ *jobs, the processing times equal to* $p = (2, 5, 6, 4, 9, 7, 3, 10, 4, 3)$ *and deadlines* $\widetilde{d} = (21, 22, 23, 24, 30, 35, 46, 49, 53, 53)$.

The minimum sequence is $(J_4, J_3, J_2, \boxed{J_1}, \boxed{J_5}, \boxed{J_6}, J_{10}, \boxed{J_7}, \boxed{J_8}, \boxed{J_9})$ *at level 36, with 6 batches.*

Following the algorithm, we first compute the jobs completion times. We obtain $C = (2, 7, 13, 17, 26, 33, 36, 46, 50, 53)$. *We put jobs* J_1 *to* J_4 *in the same batch* ($C_5 > \widetilde{d}_1$). *Then,* J_5 *and* J_6 *are alone in their batch. Then,* J_7 *and* J_8 *are in the same batch* ($C_9 > \widetilde{d}_7$) *and finally* J_9 *and* J_{10} *are in the same batch.*

We obtain sequence $(J_4, J_3, J_2, \boxed{J_1}, \boxed{J_5}, \boxed{J_6}, J_8, \boxed{J_7}, \boxed{J_{10}}, \boxed{J_9})$ *at level 39, with 5 batches.*

The algorithm runs in $O(n^2)$.

2.3 Particular cases

We consider in this section some problems with the objective function $\sum N_j$.

2.3.1 Some trivial problems: $1||\sum N_j$, $1|r_j|\sum N_j$, $1|prec|\sum N_j$

It is clear that the problem $1||\sum N_j$ is trivial. The optimal ordering of the jobs is the reverse numbering order $(J_n, J_{n-1}, ..., J_1)$ with level 0.

The problem $1|r_j|\sum N_j$ has the same optimal solution. Indeed, here, the start times of the jobs can be delayed due to the release dates, but the objective function does not depend on the completion times. Therefore, there is no problem to schedule the jobs in reverse numbering order too. Of course, introducing a common deadline would not lead to the same solution.

The problem $1|prec|\sum N_j$ can be solved to optimality by a simple backward algorithm of the same type as Alg. 8 or [Lawler, 1973]. The only modification is in the choice of the job J_r to put last: "Let $J_r \in \mathcal{J}$ such that $r = argmin\{j/J_j$ has no successor$\}$".

2.3.2 Unitary jobs

We consider in this section the case where jobs have a processing time equal to 1 and a deadline. Again, jobs are supposed to be numbered in EDD order, and at least one feasible solution exists.

2.3.2.1 Properties

Property 8 *The total number of optimal sequences of the problem* $1|\widetilde{d}_j, p_j = 1|\sum_{j=1}^{n} N_j$ *is a constant equal to:*

$$T = \prod_{j=1}^{n} \Big(\min(\widetilde{d}_j, n) - j + 1 \Big) \tag{2.10}$$

Proof
For any job J_j such that $\widetilde{d}_j \leq n$, job J_j has $M_j = \widetilde{d}_j + 1 - j$ choices for its position in the sequence.

Similarly, $\forall J_j$ such that $\widetilde{d}_j > n$, J_j has $M_j = n + 1 - j$ choices for its position.

So, the total number of optimal sequences of the problem is $T = \prod_{j=1}^{n} \Big(\min(\widetilde{d}_j, n) - j + 1 \Big)$

Example 15 *Let consider an instance with $n = 5$ jobs and the following deadlines $\tilde{d} = (3, 3, 4, 4, 5)$.*

The number of optimal sequences is equal to $T = 3 \times 2 \times 2 \times 1 \times 1 = 12$. The 12 optimal sequences are the following.

J_4	J_2	J_1	J_3	J_5	J_3	J_2	J_1	J_4	J_5
J_2	J_4	J_1	J_3	J_5	J_2	J_3	J_1	J_4	J_5
J_4	J_1	J_2	J_3	J_5	J_3	J_1	J_2	J_4	J_5
J_2	J_1	J_4	J_3	J_5	J_2	J_1	J_3	J_4	J_5
J_1	J_4	J_2	J_3	J_5	J_1	J_3	J_2	J_4	J_5
J_1	J_2	J_4	J_3	J_5	J_1	J_2	J_3	J_4	J_5

Property 9 *The minimum level of a sequence is a constant equal to:*

$$\sum_{j=1}^{n} N_j = \sum_{j=1}^{n} \max(n - \widetilde{d}_j, 0) = \sum_{j \in \{1,..,n\} / \widetilde{d}_j < n} (n - \widetilde{d}_j) \tag{2.11}$$

Proof for any job J_j such that $\widetilde{d}_j < n$, J_j is necessarily sequenced "before" $(n - \widetilde{d}_j)$ jobs having a greater index. On the contrary, for any job J_j such that $\widetilde{d}_j \geq n$, J_j has not to be sequenced "before" a job with a greater index.

The level is the number of the relations "before". So we have

$$\sum_{j=1}^{n} N_j = \sum_{j=1}^{n} \max(n - \widetilde{d}_j, 0) = \sum_{j \in \{1,..,n\}, \widetilde{d}_j < n} (n - \widetilde{d}_j)$$

Example 16 *Let consider the previous instance with $n = 5$ jobs and deadlines $\tilde{d} = (3, 3, 4, 4, 5)$.*

The EDD sequence is $\sigma^{\top} = (J_1, J_2, J_3, J_4, J_5)$ and its level is equal to 10. The sequence with minimal level is $(J_4, J_2, J_1, J_3, J_5)$ with level 6.

$Z = max(5 - 3, 0) + max(5 - 3, 0) + max(5 - 4, 0) + max(5 - 4, 0) + max(5 - 5, 0) = 2 + 2 + 1 + 1 + 0 = 6$

Remark : There is no need to have a deadline greater than n. So if we assume that $\tilde{d}_j \leq n$, $\forall j \in \{1, 2, ..., n\}$, the properties 8 and 9 become:

- The total number of optimal sequences is

$$T = \prod_{j=1}^{n} (\widetilde{d}_j - j + 1)$$

- The minimum level of a sequence is

$$\sum_{j=1}^{n} N_j = \sum_{j=1}^{n}(n - \widetilde{d}_j) = n^2 - \sum_{j=1}^{n} \widetilde{d}_j$$

2.3.2.2 Algorithms

Let remember that $\sigma^\top$ is the *root sequence* where jobs are numbered with respect to rule EDD. Sequence $\sigma^\perp$ is the sequence $(J_n, J_{n-1}, ..., J_1)$ at level 0, but not necessarily feasible with respect to the deadlines.

We search for the *minimum sequence* σ^*.

The idea of the algorithm is to take the jobs in their numbering order and to place each job before its deadline but as late as possible.

Algorithm 7 ALAP1 – As Late As Possible – $p_j = 1$

1: Input: $\widetilde{d}_j, \forall j \in \{1, 2, ..., n\}$
2: $\mathcal{P}_a \leftarrow \{1, 2, ..., n\}$ // set of available positions
3: **for** $j = 1$ to n **do**
4: Position P_j is the $(\widetilde{d}_j - (j-1))^{th}$ element of $\mathcal{P}_a$
5: $\mathcal{P}_a \leftarrow \mathcal{P}_a \backslash \{P_j\}$

The insertion and deletion of elements in a sorted array of $n-j$ element takes $O(n-j)$ time. There are n arrays with the lengths from n to 1. This algorithm runs in $O(n \log n)$ and is optimal. The proof can be done by a simple pairwise exchange argument (Alg.7 always put the job with the smallest index in the last possible position).

Example 17 *Consider again the instance of 5 jobs with* $\tilde{d} = (3, 3, 4, 4, 5)$.

We have $\mathcal{P}_a = \{1, 2, 3, 4, 5\}$.

- $j = 1$, *position* P_1 *of* J_1 *is the* $(\widetilde{d}_1 - (1-1))^{th} = 3^{rd}$ *element in* $\mathcal{P}_a \Rightarrow P_1 = 3$ *and* $\mathcal{P}_a = \{1, 2, 4, 5\}$
- $j = 2$, *position* P_2 *of* J_2 *is the* $(\widetilde{d}_2 - (2-1))^{th} = 2^{nd}$ *element in* $\mathcal{P}_a \Rightarrow P_2 = 2$ *and* $\mathcal{P}_a = \{1, 4, 5\}$
- $j = 3$, *position* P_3 *of* J_3 *is the* $(\widetilde{d}_3 - (3-1))^{th} = 2^{nd}$ *element in* $\mathcal{P}_a \Rightarrow P_3 = 4$ *and* $\mathcal{P}_a = \{1, 5\}$
- $j = 4$, *position* P_4 *of* J_4 *is the* $(\widetilde{d}_4 - (4-1))^{th} = 1^{st}$ *element in* $\mathcal{P}_a \Rightarrow P_4 = 1$ *and* $\mathcal{P}_a = \{5\}$
- $j = 5$, *position* P_5 *of* J_5 *is the* $(\widetilde{d}_5 - (5-1))^{th} = 1^{st}$ *element in* $\mathcal{P}_a \Rightarrow P_5 = 5$ *and* $\mathcal{P}_a = \emptyset$.

We introduce now a backward algorithm for solving problem $1|\widetilde{d}_j, p_j = 1| \sum_{j=1}^{n} N_j$.

Algorithm 8 BW$_{index}$ – Backward – $p_j = 1$

1: $\mathcal{J} \leftarrow \{J_1, J_2, ..., J_n\}$
2: $k \leftarrow n$
3: **while** $\mathcal{J} \neq \emptyset$ **do**
4: Let $J_r \in \mathcal{J}$ such that $r = argmin\{j \in \mathcal{J} / \tilde{d}_j \geq k\}$
5: Put J_r in position k
6: $k \leftarrow k - 1$
7: $\mathcal{J} \leftarrow \mathcal{J} \setminus \{J_r\}$

This algorithm is important because it is very intuitive (remember that such an algorithm is optimal for problems $1|prec|f_{max}$ (famous rule of [Lawler, 1973]).

Alg. 8 assigns at each position the job with the smallest index, starting by the end. Alg. 7 does the same, but placing the jobs at different positions, not in ascending or descending order. But at the end, the results are the same. The complexity of Alg. 8 is in $O(n \log n)$.

In the case where $p_j = p$, properties 9 and 8 can apply after change $\widetilde{d'}_j = \lfloor \frac{\widetilde{d}_j}{p} \rfloor$; Alg. 7 can be changed a little in order to solve the problem to optimality, Alg. 9 is optimal in this case. Similarly, Alg. 8 could be adapted by changing a little to Alg10.

Algorithm 9 ALAPp – As Late As Possible – $p_j = p$

1: Input: $\widetilde{d}_j, \forall j \in \{1, 2, ..., n\}$
2: $\mathcal{P}_a \leftarrow \{1, 2, ..., n\}$: set of available positions
3: **for** $j = 1$ to n **do**
4: Put J_j at the biggest position $P_j \in \mathcal{P}_a$ such that $p \times P_j \leq \widetilde{d}_j$
5: $\mathcal{P}_a \leftarrow \mathcal{P}_a \backslash \{P_j\}$

Algorithm 10 BWp_{index} – Backward – $p_j = p$

1: $\mathcal{J} \leftarrow \{J_1, J_2, ..., J_n\}$
2: $k \leftarrow n$
3: **while** $\mathcal{J} \neq \emptyset$ **do**
4: Let $J_r \in \mathcal{J}$ such that $r = argmin\{j \in \mathcal{J} / \tilde{d}_j \geq pk\}$
5: Put J_r in position k
6: $k \leftarrow k - 1$
7: $\mathcal{J} \leftarrow \mathcal{J} \setminus \{J_r\}$

We have the mathematical expressions for calculating the level of the minimal sequence and the number of feasible sequences, before finding the minimal sequence (Properties 9 and 8). The level of the second minimal sequence cannot be the same as the level of the first sequence. Indeed, for the first sequence at minimum level, no job can change its position without increasing the level. This problem is tackled in the next subsection.

2.3.3 Unitary jobs: next minimum sequences

As explained in Chapter 1, finding the first minimum sequence allows to characterize a lot of feasible sequences, but it is not enough if the aim is to characterize the whole set of feasible sequences.

Once a minimum sequence has been found, one has to find the other ones. We treat in this section the search of the next feasible sequences at minimum level with unitary jobs.

2.3.3.1 Finding the second minimal sequence

We are interested in the problem denoted by $1|\widetilde{d}_j, p_j = 1, 2^{nd}seq|\sum N_j$.

Let us denote by σ_1^* the 1^{st} minimum sequence which has been found. The characteristics of the predecessors of σ_1^* are of the type:

$$a_1 \prec b_1 \wedge a_2 \prec b_2 ... \wedge ... a_v \prec b_v$$

where $a_i \prec b_i$ means that $a_i < b_i$ and job J_{a_i} is before job J_{b_i}. We search for a new feasible sequence at minimum level, which is not already characterized by σ_1^*. We denote this sequence by σ_2^*.

For this sequence, we must have:

$$b_1 \prec a_1 \vee b_2 \prec a_2 ... \vee ... b_v \prec a_v$$

Example 18 *Let consider again the previous example with 5 jobs.*

Since $\sigma_1^* = (J_4, J_2, J_1, J_3, J_5)$, the characteristics of the feasible solutions characterized by σ_1^* are the following:

$$J_1 \prec J_3 \wedge J_1 \prec J_5 \wedge J_2 \prec J_3 \wedge J_2 \prec J_5 \wedge J_3 \prec J_5 \wedge J_4 \prec J_5$$

Therefore, a feasible sequence not already characterized by σ_1^* has to satisfy:

$$J_3 \prec J_1 \vee J_5 \prec J_1 \vee J_3 \prec J_2 \vee J_5 \prec J_2 \vee J_5 \prec J_3 \vee J_5 \prec J_4$$

We have some precedence relations, not linked with an "and" operator as usual, but with an "or" operator. It means that satisfying any subset of the constraints of this set is sufficient.

The problem is a special case of problem $1|\widetilde{d}_j, p_j = 1, or - prec|\sum N_j$.

We know that the level of σ_1^* is also the number of constraints, here denoted by v, and bounded by $\frac{1}{2}n(n-1)$.

The idea of the algorithm is the following. For each condition $b_i \prec a_i$ we search for a sequence with minimum level satisfying this condition. Let remember that we necessarily have $a_i < b_i$ and therefore $\widetilde{d_{a_i}} \leq \widetilde{d_{b_i}}$. To impose that J_{b_i} precedes J_{a_i}, we set its deadline to $\widetilde{d_{a_i}}$. Then, we run the equivalent of Alg. 7 and find a feasible solution at minimum level (or no feasible solution). We do this for each $i \in \{1, 2, ..., v\}$ and at the end, we keep the feasible sequence with minimum level. The algorithm ALAP2 is described in Alg. 11.

Algorithm 11 ALAP2: As Late As Possible – $p_j = 1$ – 2nd sequence

Input: σ_1^*
$S2 = \emptyset$
for i **from** 1 **to** v **do**
  $\mathcal{P}_a \leftarrow \{1, 2, ..., n\}$ // set of available positions in σ_2^i
  $\widetilde{d'}_j = \widetilde{d}_j, \forall j \in \{1, 2, ..., n\}$
  $\widetilde{d'}_{b_i} = \widetilde{d}_{a_i}$
  $j = 1$, $Continue = True$
  while $j \leq n$ **and** $Continue$ **do**
    if $\widetilde{d'}_j \leq j - 1$ **then** $Continue = False$
    else P_j is the $(\widetilde{d'}_j - (j-1))^{th}$ element of $\mathcal{P}_a$; $\mathcal{P}_a \leftarrow \mathcal{P}_a \backslash \{P_j\}$
    $j = j + 1$
  if $Continue$ **then**
    $S2 = S2 \bigcup \{\sigma_2^i\}$
return sequence $\sigma_2^i \in S2$ with minimum level

Example 19 *Let consider again the 5-job example with* $\tilde{d} = (3, 3, 4, 4, 5)$. *We consider the 6 conditions consecutively.*

- *Condition 1:* $J_3 \prec J_1$. *We now have* $\tilde{d} = (3, 3, 3, 4, 5)$. *The solution returned by Alg. 7 is* $(J_3, J_2, J_1, J_4, J_5)$ *with level 7.*
- *Condition 2:* $J_5 \prec J_1$. *There is no feasible solution in this case because the five jobs have to complete before date 4. This is the same for the conditions* $J_5 \prec J_2$, $J_5 \prec J_3$ *and* $J_5 \prec J_4$.
- *Condition 6:* $J_3 \prec J_2$. *We now have* $\tilde{d} = (3, 3, 3, 4, 5)$. *Again, the solution returned by Alg. 7 is* $(J_3, J_2, J_1, J_4, J_5)$ *with level 7.*

So the second sequence with minimum level is $\sigma_2^* = (J_3, J_2, J_1, J_4, J_5)$ *with level 7.*

There are at most $\frac{1}{2}n(n-1)$ possible values for i. Each problem can be solved in $O(n \log n)$. Therefore, problem $1|\widetilde{d}_j, p_j = 1, 2^{nd}seq|\sum N_j$ can be solved by this algorithm in $O(n^3 \log n)$.

2.3.4 The problem $1|\widetilde{d}_j, EDD = LPT|\sum N_j$

In this section, we assume that jobs have arbitrary processing times, but agreeable deadlines. More precisely we assume that EDD=LPT, in other terms:

$$\widetilde{d}_1 \leq \widetilde{d}_2 \leq ... \leq \widetilde{d}_n \bigwedge p_1 \geq p_2 \geq ... \geq p_n$$

The problem is denoted by $1|\widetilde{d}_j, EDD = LPT|\sum N_j$.

We know that Alg. 8 BW_{index} gives the exact solution for the case of *identical processing times*. We can generalize Alg.8 BW_{index} in the case of *arbitrary processing times* in the following Alg 12 BWg_{index} presented in Alg. 12.

Algorithm 12 BWg_{index} – Backward – p_j arbitrary

1: $\mathcal{J} \leftarrow \{J_1, J_2, ..., J_n\}$
2: $t \leftarrow \sum_{j=1}^{n} p_j$, $k \leftarrow n$
3: **while** $\mathcal{J} \neq \emptyset$ **do**
4: Let $J_r \in \mathcal{J}$ such that $r = argmin\{j \in \mathcal{J} / \tilde{d}_j \geq t\}$
5: Put J_r in position k
6: $t \leftarrow t - p_r$, $k \leftarrow k - 1$, $\mathcal{J} \leftarrow \mathcal{J} \setminus \{J_r\}$

Proposition 4 *The problem* $1|\widetilde{d}_j, EDD = LPT|\sum N_j$ *is solved optimally in* $O(n \log n)$ *by Alg.12* BWg_{index}.

Proof We denote by σ^{BW} the sequence obtained by Alg. 12 BWg_{index}.

Let us assume that this sequence is feasible but not at minimum level. We denote by σ^* a feasible sequence at minimum level, not returned by BWg_{index} algorithm.

We compare sequence σ^{BWg} and σ^*, starting by the end. We denote by J_k in σ^{BWg} and J_l in σ^* the jobs corresponding to the first time the two sequences differ (see Fig. 2.12).

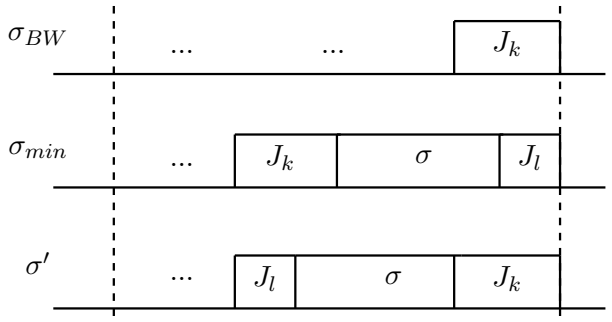

Figure 2.12: Proof of BWg_{index} in the case where $EDD = LPT$

Because of the definition of BWg_{index} algorithm, J_k is the *smallest-index job* that can be put here. Because J_k and J_l are different, we have $k < l$ and because EDD=LPT, we have $p_k \geq p_l$. Thus, we can construct a feasible sequence σ' by swapping J_k and J_l in σ^*. σ' remains feasible and because $l > k$, the level of σ' is smaller than the level of σ^*, which is not possible.

So this case is not possible and σ^{BWg} is at minimum level.

Notice that the problem $1|\widetilde{d}_j, EDD = LPT|\sum N_j$ can also be solved in $O(n \log n)$ by $FW1$ or $FW2$ algorithms, which are given in the next chapter.

2.3.5 The problem $1|\widetilde{d}_j, B = k|\sum N_j$

We refer here to the definition of batches introduced in Section 2.2.3.2. We suppose in this section that the number of batches is fixed, and we denote this constraint by $B = k$, the number of batches (as illustrated in Fig. 2.13).

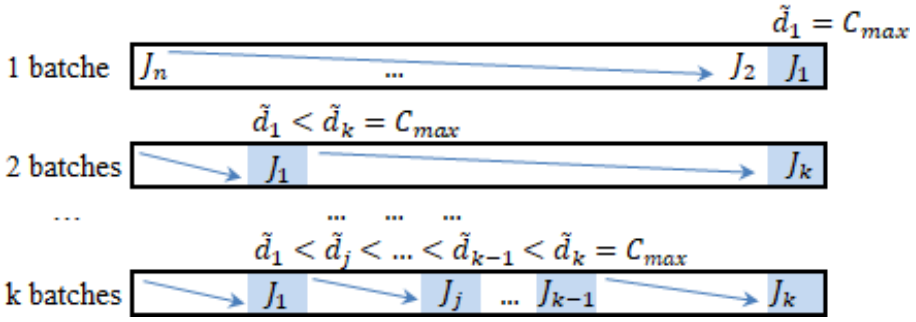

Figure 2.13: Fixed number of batches

2.3.5.1 Case $k = 1$

Suppose there is at most one batch.

The only possible solution is to put job J_1 in last position. This is possible if $\widetilde{d}_1 = \widetilde{d}_2 = ... = \widetilde{d}_n = \sum_{j=1}^{n} p_j$.

This case is trivial. We can say that if $\widetilde{d}_1 \geq \sum_{j=1}^{n} p_j$, then the optimal solution is $\sigma^{\perp} = (J_n, J_{n-1}, ..., J_1)$. Otherwise, there is no solution to the problem with only one batch.

2.3.5.2 Case $k = 2$

Suppose there are at most two batches. The first batch is composed by the jobs preceding job J_1 and the second batch is composed by the remaining jobs. We define $\mathcal{D} = \{J_j / \widetilde{d}_j < \sum_{j=1}^{n} p_j\}$.

Proposition 5 *The jobs of $\mathcal{D}$ are necessarily scheduled before J_1.*

Proof. Suppose it is not the case and denote by J_l a job of $\mathcal{D}$ which is scheduled after J_1. Then, there are more than two batches: one with the first jobs up to J_1, one with the jobs after J_1 up to J_l, and the last batch.

Therefore, if $\sum_{J_j \in \mathcal{D}} p_j > \tilde{d}_1$ there is no feasible solution with only two batches. We suppose in the following that

$$\sum_{J_j \in \mathcal{D}} p_j \leq \widetilde{d}_1 \tag{2.12}$$

Notice that the jobs in $\mathcal{D}$ have necessarily the first indices: $\mathcal{D} = \{J_1, J_2, ..., J_{|\mathcal{D}|}\}$.

The scheme of a solution with exactly two batches is given in Fig. 2.14.

By definition of a batch, the jobs before J_1 are scheduled in their reverse numbering order. It is the same for the jobs after J_1.

The expression of the problem to solve is the following. We want to find a subset of $\mathcal{J} \setminus \mathcal{D}$ jobs, called S, to put at the beginning of the schedule. The remainder jobs will be put at the end. We denote by x_j a boolean variable equal to 1 if job $J_j \in \mathcal{J} \setminus \mathcal{D}$ is put at

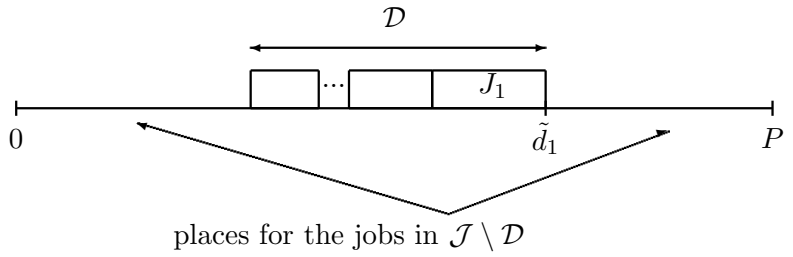

Figure 2.14: Illustration of a feasible solution when $B = 2$

the beginning of the schedule ($J_j \in S$), and 0 otherwise, for $j \in [n-N+1, n]$ if we denote by $N = |\mathcal{J} \setminus \mathcal{D}|$. The expression of the objective function, illustrated in Fig. 2.15, is the following:

$$Z = Z_1 + Z_2 \tag{2.13}$$

$$Z_1 = (n - N) \sum_{j=n-N+1}^{n} \overline{x_j} \tag{2.14}$$

$$Z_2 = \sum_{j=n-N+1}^{n} \sum_{i=j+1}^{n} x_j \overline{x_i} \tag{2.15}$$

where Z_1 (2.14) is due to the last jobs of the schedule, crossing the $n - N$ jobs of $\mathcal{D}$, and Z_2 (2.15) is the number of crossings between the jobs in $\mathcal{J} \setminus \mathcal{D}$.

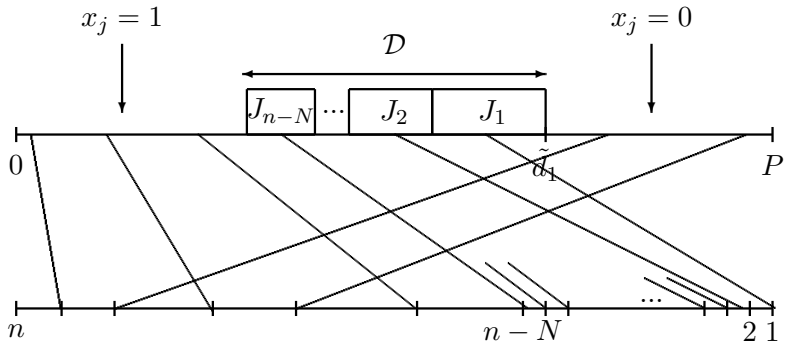

Figure 2.15: Illustration for the expression of $\sum N_j$

We have:

$$\begin{aligned}
Z &= (n-N)\sum_{j=n-N+1}^{n}\overline{x_j} + \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j\overline{x_i} \\
&= (n-N)\sum_{j=n-N+1}^{n}(1-x_j) + \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j(1-x_i) \\
&= N(n-N) - (n-N)\sum_{j=n-N+1}^{n} x_j + \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j - \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j x_i \\
&= N(n-N) - (n-N)\sum_{j=n-N+1}^{n} x_j + \sum_{j=n-N+1}^{n}(n-j)x_j - \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j x_i \\
&= N(n-N) + \sum_{j=n-N+1}^{n}(N-j)x_j - \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j x_i
\end{aligned}$$

Therefore:

$$\text{MIN } Z \Leftrightarrow \text{ MAX } Z' = \sum_{j=n-N+1}^{n}(j-N)x_j + \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j x_i$$

The only constraint is that the total duration of the jobs after $\mathcal{D}$ must exceed $\sum_{j=1}^{n} p_j - \tilde{d}_1$ (see also (2.12)), or equivalently that $\sum_{j\in S} p_j \geq \tilde{d}_1 - \sum_{j\in\mathcal{D}} p_j$. So the problem is to find x_j $(n-N+1 \leq j \leq n)$ so that:

$$\begin{aligned}
\text{MAX} \quad & \sum_{j=n-N+1}^{n}(j-N)x_j + \sum_{j=n-N+1}^{n}\sum_{i=j+1}^{n} x_j x_i \\
\text{s.t.} \quad & \sum_{j=n-N+1}^{n} p_j x_j \leq \tilde{d}_1 - \sum_{j\in\mathcal{D}} p_j \\
& x_j \in \{0,1\}, \forall j \in \{n-N+1, ..., n\}
\end{aligned}$$

Considering only the jobs in $\mathcal{J}\setminus\mathcal{D}$, this problem can be transformed into the following problem:

$$\begin{aligned}
\text{MAX} \quad & \sum_{j=1}^{n}(j+c)x_j + \sum_{j=1}^{n}\sum_{i=j+1}^{n} x_j x_i \\
\text{s.t.} \quad & \sum_{j=1}^{n} p_j x_j \leq K \\
& x_j \in \{0,1\}, \forall j \in \{1, ..., n\}
\end{aligned}$$

This problem is a 0-1 quadratic knapsack problem with the particularity that the coefficients of the quadratic term are all equal to 1. This makes the problem easier to

solve. Let suppose that the cardinality of S is fixed: $\sum_{j=1}^{n} x_j = C$. In this case, $c\sum_{j=1}^{n} x_j$ and $\sum_{j=1}^{n}\sum_{i=j+1}^{n} x_j x_i$ are constant and the problem is equivalent to:

$$\text{MAX} \sum_{j=1}^{n} j x_j \tag{2.16}$$

$$\text{s.t.} \sum_{j=1}^{n} p_j x_j \leq K \tag{2.17}$$

$$\sum_{j=1}^{n} x_j = C \tag{2.18}$$

$$x_j \in \{0,1\}, \forall j \in \{1,...,n\} \tag{2.19}$$

This problem is a "fixed cardinality 0-1 knapsack problem", for which the profit is equal to the index of the job. In terms of complexity, we can notice that the "fixed size" has no impact. Indeed, if the problem with fixed size is polynomially solvable, then the problem with "arbitrary size" is also polynomially solvable (just run the algorithm for any possible size, which is bounded by n). And for the same reasons, if the problem with arbitrary size is NP-hard, then the problem with fixed size is NP-hard as well. So we decide now to not consider the constraint of the size (2.18). To solve this problem, let us consider its dual version:

$$\text{MIN} \sum_{j=1}^{n} p_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} j x_j \geq K$$

$$x_j \in \{0,1\}, \forall j \in \{1,...,n\}$$

As we will see, it is possible to derive a Dynamic Programming algorithm for solving this problem. The general expression of the dual problem is:

$$\text{MIN} \sum_{j=1}^{n} p_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_j x_j \geq K$$

A DP algorithm can be proposed with a time complexity in $O(nK)$. We define $F_j(k)$ the minimum duration of a subset of $\{J_1, J_2, ..., J_j\}$ having a reward at least equal to K. And we have:

$$F_0(k) = \infty, \forall k$$

$$F_j(k) = 0, \forall k \leq 0, \forall j$$

$$F_j(k) = \begin{cases} \min\left(F_{j-1}(k), F_{j-1}(k-a_j) + p_j\right), \forall j \in \{1,...,n\}, \forall k \in \{a_j,...,K\} \\ F_{j-1}(k-1), \forall j \in \{1,...,n\}, \forall k \in \{0,...,a_j - 1\} \end{cases}$$

The optimal value is given by $F_n(K)$ and the solution is obtained by a simple backward algorithm.

In our case, $a_j = j$, $\forall j \in \{1, ..., n\}$, therefore $\sum_{j=1}^{n} a_j$ is bounded by $\frac{n(n+1)}{2}$, and thus the algorithm runs in polynomial time in $O(n^3)$.

Example 20 *Let consider an instance with 6 jobs. We assume that there is a job J_1 with a duration equal to 11 and a deadline equal to $\tilde{d}_1 = 43$. The data of the other jobs are given in the following table (to feet with the DP algorithm the jobs are renumbered from 1 to 5 and the reward a_j is kept to the initial value, i.e. $j+1$). The deadlines of the other jobs are equal to 75.*

j	*1*	*2*	*3*	*4*	*5*
a_j *(job number)*	*2*	*3*	*4*	*5*	*6*
p_j	*12*	*11*	*16*	*14*	*11*

We have $\sum_{j=1}^{n} p_j = 64$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
1	0	0	12	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2	0	0	11	11	11	23	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
3	0	0	11	11	11	16	27	27	27	**39**	–	–	–	–	–	–	–	–	–	–	–
4	0	0	11	11	11	14	14	25	25	25	30	41	41	41	53	–	–	–	–	–	–
5	0	0	11	11	11	11	11	11	22	22	22	25	25	36	36	36	41	52	52	52	64

Table 2.3: Details of the DP algorithm for the dual problem ($-$ stands for ∞)

We want the minimum possible value of the sum of indices (k) so that the total duration is greater than or equal to 32. These jobs will be put after job J_1 in the solution (leading to a partition in two sets of equal length). In Table 2.3, we search for the column with the smallest index, in which there is a number greater than or equal to 32. It is column 9, line 3, for $F_3(9) = 39$. This value comes from $F_2(9-4) + 16 = 39$, therefore we take job J_3. The value of $F_2(5)$ comes from $F_2(5-3) + 11 = 23$, so we take job J_2. Finally, we also take job J_1 and we have three jobs J_3, J_2, J_1 with a duration greater than or equal to 35, with the smallest sum of indices. The corresponding jobs J_4, J_3 and J_2 are put after the "real" job J_1 in this order, and the first jobs are J_6 and J_5 in this order. Sequence $(J_6, J_5, J_1, J_4, J_3, J_2)$ is optimal.

Last jobs numbered in LPT order We consider the case where the jobs in $\mathcal{J} \setminus \mathcal{D}$ (which have the same deadline) are numbered in decreasing processing time order (LPT order). We denote by d the number of jobs in $\mathcal{D}$.

We consider the following algorithm called GR (Greedy). We put the jobs of $\mathcal{J} \setminus \mathcal{D}$ in their reverse numbering order: $S = (J_n, J_{n-1}, ..., J_{d+1})$. Then, we insert the jobs of $\mathcal{D}$ consecutively in their reverse numbering order at the last possible position so that $S = (J_n, J_{n-1}, ..., J_k, (J_d, J_{d-1}, ..., J_1), J_{k-1}, ...J_{d+1})$. The algorithm 13 GR describes the method, which runs in $O(n)$.

Algorithm 13 GR

1: $t = 0$, $k = n$, $S \leftarrow \emptyset$
2: **while** $(t + p_k + \sum_{J_j \in \mathcal{D}} p_j \leq \widetilde{d}_1)$ and $(k \geq d + 1)$ **do**
3: $\quad S \leftarrow S + J_k$
4: $\quad t = t + p_k$, $k \leftarrow k - 1$
5: $S \leftarrow S + (J_d, J_{d-1}, ..., J_1) + (J_{k-1}, J_{k-2}, ..., J_{d+1})$
6: **return** S

Example 21 *We consider an instance with $n = 8$ jobs, processing times $p = (7, 5, 3, 9, 5, 4, 3, 2)$ and deadlines $\tilde{d} = (28, 29, 30, 38, 38, 38, 38, 38)$.*

We schedule J_8, then J_7 and J_6. Here, we have to schedule (J_3, J_2, J_1) so that J_1 completes at time 24. Then we terminate with job J_5 and J_4, which constitute the second batch. The final solution is $(J_8, J_7, J_6, J_3, J_2, J_1, J_5, J_4)$ and $\sum N_j = 6$.

Proposition 6 *Algorithm 13 GR is optimal for problem $1|\widetilde{d}_j, B = 2, LPT|\sum N_j$. Moreover, $\sum N_j = d.r$, where d, r are the number of jobs in $\mathcal{D}$ and after $\mathcal{D}$, respectively.*

Proof. See Appendix 1.

2.4 Conclusion

In this chapter, we present the main theory, conceptions, mathematical expressions and some results for particular cases for finding a sequence minimizing a new objective function denoted $\sum N_j$, corresponding to a distance to a reference sequence.

We explain the relations between $\sum N_j$ and many other notions, such as Kendall's-τ distance, the crossing number in permutations, the One Sided Crossing Minimization problem and the Checkpoint Ordering Problem. These problems have been the subject of much research and many interesting results exist for theoretical and practical interests.

Mathematical Programming formulations of the objective function are presented, with positional variables and linear ordering variables. Some properties show a relation between $\sum N_j$ and $\sum jP_j$ or $\sum P_j$. This new type of objective function never appeared in the scheduling literature before, to the best of our knowledge.

The first results which are obtained for this objective function are summarized in the following table.

Problem	Complexity	
$1\|\|\sum N_j$	$O(1)$	solution $\sigma^{\perp}$ is optimal
$1\|r_j\|\sum N_j$	$O(1)$	solution $\sigma^{\perp}$ is optimal
$1\|prec\|\sum N_j$	$O(n\log n)$	Backward algorithm
$1\|p_j=1,\widetilde{d}_j\|\sum N_j$	$O(n\log n)$	BW and ALAP1 algorithm
$1\|\widetilde{d}_j,p_j=1,2^{nd}seq\|\sum N_j$	$O(n^3\log n)$	Iterated BW algorithm or Iterated ALAP1 algorithm
$1\|EDD=LPT,\widetilde{d}_j\|\sum N_j$	$O(n\log n)$	BW, FW1 or FW2 algorithm
$1\|\widetilde{d}_j,B=1\|\sum N_j$	$O(1)$	Simple test to know if $\sigma^{\perp}$ is feasible
$1\|\widetilde{d}_j,B=2\|\sum N_j$	$O(n^3)$	Dynamic programming
$1\|\widetilde{d}_j,B=2,LPT\|\sum N_j$	$O(n)$	Algorithm GR
$1\|\widetilde{d}_j\|\sum N_j$	open	

Chapter 3

Resolution methods for finding a minimum sequence

In this chapter, we present some resolution methods for problem $1|\tilde{d}_j|\sum N_j$. The resolution methods are presented in two parts: the non-polynomial time methods and the polynomial-time algorithms. The chapter terminates by some computational experiments. Remember that the complexity of this problem is open.

3.1 Non-polynomial time methods

In this section, we present mathematical programming formulations for the $1|\widetilde{d}_j|\sum N_j$ problem, a dynamic programming algorithm and a branch-and-bound algorithm.

3.1.1 Mathematical programming formulations for problem $1|\widetilde{d}_j|\sum N_j$

In this section we propose several Mixed Integer Linear Programming models.

3.1.1.1 MILP1: Positional variables

We use in this model positional variables defined in Section 1.1.3.1:

$x_{j,k} = \begin{cases} 1 & \text{if job } J_j \text{ is in position } k \\ 0 & \text{otherwise} \end{cases}$

The processing time and the deadline of the job at position k become the variables equal to $\sum_{k=1}^{n} p_j x_{j,k}$ and $\sum_{k=1}^{n} \widetilde{d}_j x_{j,k}$ respectively.

The completion time of the job in position k is given by $C_{[k]} = \sum_{q=1}^{k}\sum_{j=1}^{n} p_j x_{j,q}$

We introduce variables $n_{j,k}$ equal to the number of jobs after position k with an index

greater than j. So, in some sense, we have:

$$n_{j,k} = \sum_{j'=j+1}^{n} \sum_{k'=k+1}^{n} x_{j',k'} \tag{3.1}$$

but this quantity has to be counted only if variable $x_{j,k}$ is equal to 1. This leads to a nonlinear expression which is linearized in the model through big-M constraints.

We have $N_j = \sum_{k=1}^{n} n_{j,k}$ the contribution of job J_j to the objective function.

The whole model is the following.

$$\text{MIN} \sum_{j=1}^{n} N_j \tag{3.2}$$

$$\text{s.t. } (1.1), (1.2) \tag{3.3}$$

$$\sum_{j'=j+1}^{n} \sum_{k'=k+1}^{n} x_{j',k'} - HV(1 - x_{j,k}) \leq n_{j,k}, \forall j \in \{1, 2, ...n-1\}, \forall k \in \{1, 2, ...n-1\} \tag{3.4}$$

$$\sum_{q=1}^{k} \sum_{j=1}^{n} p_j x_{j,q} \leq \sum_{j=1}^{n} \widetilde{d}_j x_{j,k}, \forall k \in \{1, 2, ...n\} \tag{3.5}$$

$$N_j = \sum_{k=1}^{n} n_{j,k}, \forall j \in \{1, 2, ...n\} \tag{3.6}$$

The model contains four sets of constraints. Equations (3.4) allow to compute variables $n_{j,k}$: N_j takes the value of equation (3.1) only if $x_{j,k}$ is equal to 1. Finally, constraints (3.5) guaranty the respect of the deadlines.

This model contains n^2 binary variables, n^2 continuous variables and $3n+n(n-1)/2+1$ constraints, among them, $n(n-1)/2$ are big-M constraints. notice that the use of variables N_j (3.6) is not mandatory.

In practice, we can introduce the two following cuts:

$$n_{j,k} < n \tag{3.7}$$

$$\sum_{j=1}^{n} N_j \leq n(n-1)/2 \tag{3.8}$$

3.1.1.2 MILP2: Linear Ordering variables

We define variables of relative positions as in Section 1.1.3.1.

$$y_{i,j} = \begin{cases} 1 & \text{if job } J_i \text{ precedes } J_j \\ 0 & \text{otherwise} \end{cases}$$

In this case, we generally introduce a continuous variable C_j for the completion time of job J_j and the expression of the disjunctive constraint (only one job at a time on a

machine) requires a big-M as follows:

$$C_j \geq C_i + p_j - M(1 - y_{i,j}), \forall i \in \{1, 2, ...n\}, \forall j \in \{1, 2, ...n\} \quad (3.9)$$
$$C_i \geq C_j + p_i - M y_{i,j}, \forall i \in \{1, 2, ...n\}, \forall j \in \{1, 2, ...n\} \quad (3.10)$$

If J_i precedes J_j, $y_{i,j} = 1$. In this case, the (3.9) constraints ensure that $C_j \geq C_i + p_j$ and the (3.10) constraints are deleted because C_i s always greater than $C_j + p_i - M$.

If J_j precedes J_i, $y_{i,j} = 0$. In this case, the (3.9) constraints can be deleted because C_j is always greater than $C_i + p_j - M$ and the (3.10) constraints ensure that $C_i \geq C_j + p_i$.

These variables have to satisfy the triangle inequalities (3.12). The respect of the deadlines is simply given by constraints (3.14).

The whole model is the following:

$$\text{MIN} \sum_{i=1}^{n} \sum_{j=i+1}^{n} y_{i,j} \quad (3.11)$$
$$\text{s.t. } y_{i,k} + 1 \geq y_{i,j} + y_{j,k}, \forall i \in \{1, 2, ...n\}, \forall j \in \{1, 2, ...n\}, \forall k \in \{1, 2, ...n\} \quad (3.12)$$
$$(3.9), (3.10) \quad (3.13)$$
$$C_j \leq \tilde{d}_j, \forall j \in \{1, 2, ...n\} \quad (3.14)$$

This model contains n^2 binary variables, n continuous variables and $O(n^3)$ constraints, among them $2n^2$ are "big-M constraints".

It is possible to improve this model.

MILP3: Linear Ordering variables There is a more simple way to express the completion time of job J_j:

$$C_j = \sum_{i=1}^{n} p_i y_{i,j} + p_j$$

This leads to the following model MILP3:

$$\text{MIN} \sum_{i=1}^{n} \sum_{j=i+1}^{n} y_{i,j} \quad (3.15)$$
$$\text{s.t. } y_{j,k} + y_{k,l} + y_{l,j} \leq 2, \forall j \in \{1, 2, ...n\}, \forall k \in \{1, 2, ...n\}, \forall l \in \{1, 2, ...n\}, j \neq k \neq l \quad (3.16)$$
$$y_{i,j} + y_{j,i} = 1, \forall i \in \{1, 2, ...n\}, \forall j \in \{1, 2, ...n\}, \quad (3.17)$$
$$\sum_{i=1}^{n} p_i y_{i,j} + p_j \leq \tilde{d}_j, \forall j \in \{1, 2, ...n\} \quad (3.18)$$

This model contains $O(n^3)$ constraints, but no "big-M" constraints.

Remark: Model MILP3 is similar to the models for linear ordering problems. A linear ordering problem (should be presented before) can be modeled by using variables $u_{i,j} = 1$ if $i \prec j$ and 0 otherwise. The linear ordering problem can be modeled as follows:

$$\begin{aligned} MAX & \sum_{i \in V_n} \sum_{j \in V_n, j \neq i} w_{i,j} u_{i,j} \\ \text{s.t. } & u_{i,j} + u_{j,i} = 1, \forall i, j \in v_n, i \neq j \\ & u_{i,j} + u_{j,k} + u_{k,i} \leq 2, \forall i, j, k \in V_n, i \neq j \neq k \end{aligned}$$

In our problem, the weights are very particular, because $w_{i,j} = 1$ if $i > j$ and 0 otherwise. So, without considering the respect of the deadlines, our problem has a trivial solution, where the nodes are sorted in decreasing numbering order.

3.1.1.3 Finding the second minimal sequence

In this section, we are interested in the integer programming formulation of the problem of finding the second minimal sequence, by using the MILP models.

Of course, the models differs, depending on the variables definition.

Model with position variables Suppose that we have positional variables $x_{j,k}$.

Suppose that the optimal feasible sequence σ_1^* has been found and the characterization gives:

$$a_1 \prec b_1 \wedge a_2 \prec b_2 ... \wedge ... a_v \prec b_v$$

meaning that a_i is before b_i and a_i is smaller than b_i.

We search for a new sequence such that:

$$b_1 \prec a_1 \vee b_2 \prec a_2 ... \vee ... b_v \prec a_v$$

For each couple (a_i, b_i) we introduce a binary variable $w_{1,i}$ equal to 1 if $b_i \prec a_i$. Then, we introduce the following constraints.

$$\sum_{k=1}^{n} k x_{b_i,k} \leq \sum_{k=1}^{n} k x_{a_i,k} + M(1 - w_{1,i}), \forall i \in \{1, 2, ..., v\} \tag{3.19}$$

$$\sum_{h=1}^{v} w_{1,h} \geq 1 \tag{3.20}$$

Constraints (3.19) ensure that if $w_{1,i} = 1$ then $b_i \prec a_i$, and constraints (3.20) ensure that at least one of these constraints is respected.

Adding these constraints to MILP1 allows to find a second feasible sequence at minimum level not characterized by the first minimum sequence.

Notice that this process generates v new binary variables and v big-M constraints. Notice also that this process can be iterated for finding the third not characterized feasible sequence, etc.

Model with linear ordering variables Suppose that we have the relative variables $y_{i,j}$.

We need to introduce the following constraints:

$$\sum_{h=1}^{v} y_{b_h,a_h} \geq 1 \tag{3.21}$$

Constraints (3.21) ensure that at least one of the constraints $b_h \prec a_h$, $h \in \{1,..,v\}$ is respected.

Adding the constraint (3.19) to MILP2 or MILP3 allows to find a second feasible sequence at minimum level not characterized by the first minimum sequence.

Notice that this process generates only one sequence, and no additional variable. It can be iterated for finding the third not characterized feasible sequence.

3.1.2 Dynamic Programming formulation

We can propose a very basic DP algorithm for solving the problem.

Let us call $\mathcal{J}$ a group of jobs ending by J_j, $k = |\mathcal{J}|$ is the number of jobs in $\mathcal{J}$.

We define $K_j(\mathcal{J},k)$ equal to the number of jobs in $\{J_1,J_2,..,J_n\} \setminus \mathcal{J}$ with an index greater than j if $C_j \leq \widetilde{d}_j$, and ∞ otherwise.

We call $K_j(\mathcal{J},k)$ = the cost of J_j in position k after $\mathcal{J} \setminus \{J_j\}$, or more simply the contribution of job J_j at this place, to the objective function.

We have the following recursive relation:

$$F_k(\mathcal{J}) = \min_{j\in\mathcal{J}} \big(F_{k-1}(\mathcal{J}\backslash\{J_j\}) + K_j(\mathcal{J},k)\big) \tag{3.22}$$

The initial condition is $F_0(\emptyset) = 0$ and we search for $F_n(\{J_1,J_2,...,J_n\})$. Because k takes its values in $(1,2,...,n)$, this DP algorithm has an exponential time complexity.

Example 22 *We consider a 4-job example with* $p = (1,2,5,4)$ *and* $\tilde{d} = (8,9,12,12)$.

k = 0: $F_0(\emptyset) = 0$

k = 1: *all the jobs can be put in position 1.*

$\mathcal{J}$	$\{J_1\}$	$\{J_2\}$	$\{J_3\}$	$\{J_4\}$
j	*1*	*2*	*3*	*4*
$K_j(\mathcal{J},1)$	*3*	*2*	*1*	*0*
$F_1(\mathcal{J})$	***3***	***2***	***1***	***0***

k = 2: $F_2(\mathcal{J}) = \min_{j\in\mathcal{J}} \big(F_1(\mathcal{J} \setminus \{J_j\}) + K_j(\mathcal{J},2)\big)$

$\mathcal{J}$	$\{J_1,J_2\}$		$\{J_1,J_3\}$		$\{J_1,J_4\}$		$\{J_2,J_3\}$		$\{J_2,J_4\}$		$\{J_3,J_4\}$	
j	*1*	*2*	*1*	*3*	*1*	*4*	*2*	*3*	*2*	*4*	*3*	*4*
$F_1(\mathcal{J}\setminus\{J_j\})$	*2*	*3*	*1*	*3*	*0*	*3*	*1*	*2*	*0*	*2*	*0*	*1*
$K_j(\mathcal{J},2)$	*2*	*2*	*2*	*1*	*2*	*0*	*1*	*1*	*1*	*0*	*0*	*0*
$F_1(\mathcal{J}\setminus\{J_j\})+K_j(\mathcal{J},2)$	*4*	*5*	*3*	*4*	*2*	*3*	*2*	*3*	*1*	*2*	*0*	*1*
$F_2(\mathcal{J})$	*4*		*3*		*2*		*2*		*1*		*0*	

k = 3: $F_3(\mathcal{J}) = min_{j\in\mathcal{J}}(F_2(\mathcal{J}\setminus\{J_j\})+K_j(\mathcal{J},3))$

$\mathcal{J}$	$\{J_1,J_2,J_3\}$			$\{J_1,J_2,J_4\}$			$\{J_1,J_3,J_4\}$			$\{J_2,J_3,J_4\}$		
j	*1*	*2*	*3*	*1*	*2*	*4*	*1*	*3*	*4*	*2*	*3*	*4*
$F_2(\mathcal{J}\setminus\{J_j\})$	*2*	*3*	*4*	*1*	*2*	*4*	*0*	*2*	*3*	*0*	*1*	*2*
$K_j(\mathcal{J},3)$	*1*	*1*	*1*	*1*	*1*	*0*	∞	*0*	*0*	∞	*0*	*0*
$F_2(\mathcal{J}\setminus\{J_j\})+K_j(\mathcal{J},3)$	*3*	*4*	*5*	*2*	*3*	*4*	∞	*2*	*3*	∞	*1*	*2*
$F_3(\mathcal{J})$	*3*			*2*			*2*			*1*		

k = 4: $F_4(\mathcal{J}) = \min_{j\in\mathcal{J}}(F_3(\mathcal{J}\setminus\{J_j\})+K_j(J,4))$

J	$\{J_1,J_2,J_3,J_4\}$			
j	*1*	*2*	*3*	*4*
$F_3(\mathcal{J}\setminus\{J_j\})$	*1*	*2*	*2*	*3*
$K_j(\mathcal{J},4)$	∞	∞	*0*	*0*
$F_3(\mathcal{J}\setminus\{J_j\})+K_j(\mathcal{J},4)$	∞	∞	***2***	***3***
$F_4(\mathcal{J})$	*2*			

Result:

- *With $\mathcal{J}=\{J_1,J_2,J_3,J_4\}$, $F_4(\mathcal{J})=2$ corresponding to $j=3$. Therefore, J_3 is at position 4.*
- *With $\mathcal{J}=\{J_1,J_2,J_4\}$, $F_3(\mathcal{J})=2$ corresponding to $j=1$, therefore J_1 is at the position 3.*
- *With $\mathcal{J}=\{J_2,J_4\}$, $F_2(\mathcal{J})=1$ corresponding to $j=2$, therefore J_2 is at position 2.*
- *Finally, with $J=\{J_4\}$, we have J_4 at position 1.*

If we have n jobs, we have to enumerate all parts of a set of size n, and there are 2^n parts of a set. So the complexity of this DP algorithm is in $O(2^n)$.

3.1.3 Branch-and-Bound

The $B\&B$ method for problem $1|\tilde{d}_j|\sum N_j$ has the following characteristics.

The **initial upper bound** is given by the best polynomial heuristic method (Backward, Forward, see Section 3.2).

The strategy of branching is the following:

- Consider first the initial set of unscheduled jobs in reverse numbering order $S = \{J_n, J_{n-1}, ..., J_1\}$.
- Build the schedule starting by the end, at each level, add a new unscheduled job J_j in first place of σ, at the end of the schedule.

While $S \neq \emptyset$, the job J_j is chosen from the right to the left of S (i.e. from smaller index to bigger index) so that:

- the deadlines are always respected (put only in the first position of sequence σ a non tardy job).
- the dominance conditions are respected: keep only the sequences satisfying the Property 6 of batches and others properties, according to Alg. 14.

Algorithm 14 Check-head(σ, k, h)

Input:
• σ a node of scheduled jobs ($\sigma_{[1]}$ the index of the first job in $\sigma_{[1]}$),
• k the index of the next job to schedule (before σ),
• h the index of the head of the current batch
$ok = 0$
if $(k > \sigma_{[1]})$ and $(k > h)$ **then**
  $ok = 1$
else if $(k < \sigma_{[1]})$ and $(k > h)$ **then**
  $ok = 0$
else if $k < h$ **then**
  $ok = 2$, $h = k$
return (ok)

This algorithm studies three possible cases:

1. If $(k > \sigma_{[1]})$ and $(k > h)$, then there is no problem, job J_k is put before $J_{\sigma_{[1]}}$ and belongs to the same batch,
2. If $(k < \sigma_{[1]})$ and $(k > h)$, then this solution is dominated. There is no interest to put J_k before $J_{\sigma_{[1]}}$ in the batch,
3. If $k < h$, then there is no problem, job J_k is the first job of a new batch.

The **Lower Bound** can be determined in two ways:

- First method: complete the beginning of the schedule by the unscheduled jobs in reverse numbering order and evaluate the corresponding full sequence (can be done in $O(n^2)$ time).
- Second method: Consider the following definition of the positions of the jobs in S:

$S = \{J_{a_\nu}, ..., J_{a_k}, ..., J_{a_1}\}$	J_{a_ν}	$J_{a_{\nu-1}}$	...	J_{a_k}	...	J_{a_2}	J_{a_1}
position	$\nu - 1$	$\nu - 2$	...	$\nu - k$	...	1	0

Then:

$$LB(J_{a_k} + \sigma) = LB(\sigma) + (k-1)$$

This expression can be computed in $O(1)$ time.

We denote by N a node in the B&B algorithm. A node N is composed by a tuple (S, σ, lb, t, i) with S the unscheduled jobs, σ the current sequence, lb the lower bound of the node, t the starting time of σ and i the index of the head of the current batch. σ^* denotes the best current sequence. Notice that when the value of t of the current node is greater than or equal to $\widetilde{d}_1$, we immediately can generate a leaf node with the schedule starting by the unscheduled jobs in their reverse numbering order. The notation $InverseS(N)$ is the reverse order of sequence $S(N)$. We have the following algorithm Alg. 15.

Algorithm 15 B&B - $\sum N_j$

$S = \{J_n, J_{n-1}, ..., J_1\}$, $\sigma = \emptyset$, $lb = 0$, $t = \sum_{j=1}^{n} p_j$, $i = n+1$
$N = (S, \sigma, lb, t, i)$
$\pi =$ sequence returned by a heuristic, $\sigma^* = \pi$, $UB = \sum N_j(\pi)$
$Q \leftarrow N$
while $Q \neq \emptyset$ **do**
    $N = Q[1]$
    if $|\sigma(N)| = n$ **then**
        if $lb(N) < UB$ **then** Update UB and σ^*
    else
        for $J_{a_k} \in InverseS(N)$ **do** // from smallest index to biggest index
            if $a_k = 1$ **then**
                $NewN \leftarrow (\emptyset, S(N) + \sigma(N), lb(N), 0, 1)$, $Q \leftarrow NewN$
            else
                if Check-head$(\sigma(N), a_k, i) = 2$ **then**
                    $NewN = (S(N) \setminus \{J_{a_k}\}, J_{a_k} + \sigma(N), lb(N) + k - 1, t(N) - p_{a_k}, a_k)$
                    if $lb(NewN) < UB$ **then** $Q \leftarrow NewN$
                if Check-head$(\sigma(N), k, i) = 1$ **then**
                    $NewN = (S(N) \setminus \{J_k\}, J_k + \sigma(N), lb(N) + k - 1, t(N) - p_k, i(N))$
                    if $lb(NewN) < UB$ **then** $Q \leftarrow NewN$
return σ^*

Example 23 *Consider $n = 5$ jobs with processing times $p = \{2, 2, 3, 6, 8\}$ and deadlines $\widetilde{d} = \{8, 9, 16, 18, 21\}$.*

At the beginning, the root node is $N1 = (\{J_5, J_4, J_3, J_2, J_1\}, \emptyset, 0, 21, 6)$.

The heuristic returns sequence $\pi = (J_2 J_1 J_4 J_3 J_5)$ and $UB = \sum N_j(\pi) = 8$.

To respected of deadline, only job J_5 can be scheduled in the last position. So we generate only one child node $N2$ with:

- $S(N2) = S(N1) \setminus \{J_{a_k}\} = \{J_4, J_3, J_2, J_1\}$,
- $\sigma(N2) = (J_5)$,

- $lb(N2) = 4$, *the position of* J_5 *in* $S(N1)$,
- $t = 21 - p_5 = 13$,
- $i = 5$, *the new head.*

We obtain $N2 = (\{J_4, J_3, J_2, J_1\}, (J_5), 4, 13, 5)$.

From $N2$ *we can generate 2 child nodes: one with* J_3 *in last position and one with* J_4. *We start by generating the child node with* J_3*:*

- $S(N3) = S(N2) \setminus \{J_{a_k}\} = \{J_4, J_2, J_1\}$,
- $\sigma(N3) = (J_3, J_5)$,
- $lb(N3) = 6$, *the position of* J_3 *in* $S(N2)$ *is 2,*
- $t = 13 - 3 = 10$
- $i = 3$, *the new head.*

We obtain $N3 = (\{J_4, J_2, J_1\}, (J_3, J_5), 6, 10, 3)$.

We generate the child node $N4$ *with* J_4*:*

- $S(N4) = S(N2) \setminus \{J_{a_k}\} = \{J_3, J_2, J_1\}$,
- $\sigma(N4) = (J_4, J_5)$,
- $lb(N4) = 7$, *the position of* J_4 *in* $S(N2)$ *is 3,*
- $t = 13 - 6 = 7$
- $i = 4$, *the new head.*

We obtain $N4 = (\{J_3, J_2, J_1\}, (J_4, J_5), 7, 7, 4)$.

We explore now node $N3$. *Because* $t(N3) = 10$, *the only possible job is* J_4. *We generate the child node* $N5$*:*

- $S(N5) = S(N3) \setminus \{J_{a_k}\} = \{J_2, J_1\}$,
- $\sigma(N5) = (J_4, J_3, J_5)$,
- $lb(N5) = 8$, *the position of* J_4 *in* $S(N3)$ *is 2,*
- $t = 10 - 6 = 4$
- $i = 3$, *unchanged.*

We obtain $N5 = (\{J_2, J_1\}, (J_4, J_3, J_5), 8, 4, 3)$. *Because* $lb(N5) = UB$, *we do not continue the exploration of this node.*

We explore now node $N4$. *Because* $t(N4) = 7$, J_1 *can be scheduled last, and therefore generate the following leaf node:*

- $S(N6) = \emptyset$,
- $\sigma(N6) = (J_3, J_2, J_1, J_4, J_5)$,
- $lb(N4) = 7$,
- $t = 0$
- $i = 1$, *the new head.*

We obtain $N6 = (\emptyset, (J_3, J_2, J_1, J_4, J_5), 7, 0, 1)$. *The evaluation of this node is equal to 7, and it is the optimal solution.*

This example is illustrated in Fig. 3.1.

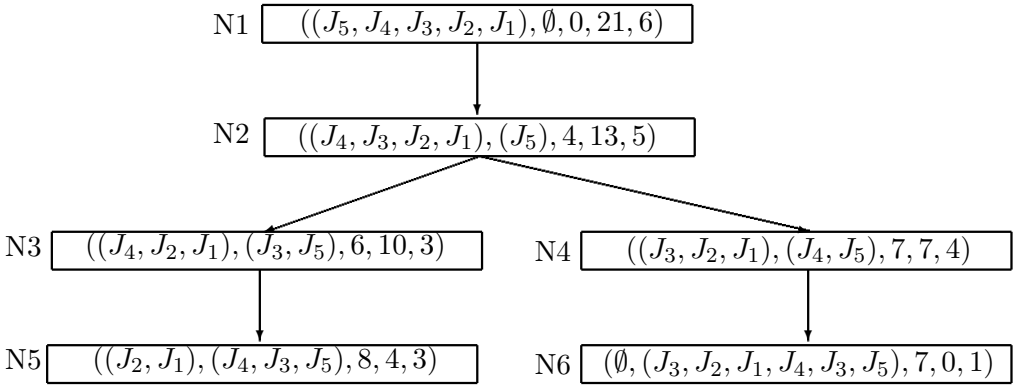

Figure 3.1: Illustration of the B&B algorithm

3.2 Polynomial time heuristics

We present in this section some polynomial time heuristic algorithms.

3.2.1 Backward algorithm

We have seen in Section 2.3.2.2 that the backward algorithm Alg. 8 was optimal in the case of unit processing times. A more general algorithm of 8 is Alg 12, an exact method for the case $EDD = SPT$. Moreover, Alg 12 becomes an interesting polynomial heuristic method for the case *arbitrary processing time.*

Example 24 *Consider an instance with* $n = 6$ *jobs and* $p = (3, 8, 4, 2, 3, 5)$ *and* $\widetilde{d} = (15, 16, 22, 25, 25, 25)$

At the beginning, $t = \sum_{j=1}^{n} p_j = 25$.

The smallest index of jobs with a deadline greater than or equal to $t = 25$ *is* J_4. *So* J_4 *is put in position 6.* t *is now equal to 23. The next job with the smallest index and a*

deadline greater than or equal to t is J_5, which is put in position 5. Then, comes J_3, J_2, J_1 and J_6.

The final sequence is $(J_6, J_1, J_2, J_3, J_5, J_4)$ with a level equal to 9.

Example 25 *Consider a second example with $n = 10$ jobs. The processing times are equal to $p = (1, 1, 1, 35, 3, 99, 74, 5, 13, 3)$ and the deadlines are equal to $\widetilde{d} = (78, 156, 234, 235, 235, 235, 235, 235, 235, 235)$. At the beginning, $t = \sum_{j=1}^{n} p_j = 235$. The smallest index of the jobs that can be put in last position is J_4. Then t is equal to 200. The smallest index of jobs is J_3, then come J_5 and J_6. At this moment, $t = 97$ and we put job J_2. We continue with job J_7, job J_1 and finally jobs J_8 to J_{10}.*

The final sequence is $(J_{10}, J_9, J_8, J_1, J_7, J_2, J_6, J_5, J_3, J_4)$ with a level equal to 11.

Worst case analysis We do now a worst case analysis of this algorithm. Let consider the following instance, with $m + 2$ jobs and the following durations and deadlines.

j	1	2	3	...	$m-2$	$m-1$	m	$m+1$	$m+2$
p_j	2	2	2	...	2	2	2	$2m$	2
$\tilde{d}_j$	$2m+2$	$2m+4$	$2m+6$	...	$4m-4$	$4m-2$	$4m$	$4m+1$	$4m+2$

Starting at $t = 2 \times (m + 1) + 2m = 4m + 2$, the only possible job to schedule at the end is job J_{m+2}. Then, at time $4m$, we have the choice between J_m and J_{m+1}. We select job J_m because it has the smallest index.

Then, at time $4m-2$ we will select J_{m-1}, etc. At the end, Alg. 12 returns the sequence: $(J_{m+1}, J_1, J_2, J_3, ..., J_{m-2}, J_{m-1}, J_m, J_{m+2})$. The level of this sequence is equal to

$$Lev^{BW_{index}} = \frac{(m+1)(m+2)}{2} + 1$$

If we consider that $n = m + 2$ is the number of jobs, we have

$$Lev^{BW_{index}} = \frac{n(n-1)}{2} + 1$$

However, the sequence with minimum level is $(J_m, J_{m-1}, J_{m-2}, ..., J_3, J_2, J_1, J_{m+1}, J_{m+2})$, with a level equal to

$$Lev^* = 2m + 1 = 2n - 3$$

The ratio $Lev^{BW_{index}}/Lev^*$ is equal to:

$$\begin{aligned} \frac{Lev^{BW_{index}}}{Lev^*} &= \frac{\frac{(n)(n-1)}{2} + 1}{2n-3} \\ &= \frac{n^2 - n + 2}{4n - 6} = \frac{1}{4}(n + \frac{1}{2}) \end{aligned}$$

Therefore the worst possible ratio of Alg. 12 is not a constant, but linear in n.

3.2.2 Forward algorithms

Inspired by Alg. ALAP1 (7) in the case of unit processing times, we construct two Forward Algorithms with the following main ideas: take the jobs in EDD order; put each job as late as possible and insert the feasible and most suited jobs before it.

We denote by $J_{[k]}$ the job in position k and $P_j(\sigma)$ denotes the position of J_j in sequence σ. We assume that σ is a sequence of type $\sigma_1 J_{[si]} \sigma_2 J_k \sigma_3$ with σ_1, σ_2 and σ_3 three partial sequences. The notation $(\sigma_1 J_k)$ indicates that J_k is inserted in σ_1 at the best position, so that σ_1 plus J_k is sorted in non increasing index order.

Algorithm 16 FW1$_{index}$ - $\sum N_j$

1: $\sigma = (J_1, J_2, ..., J_n)$
2: $si = 1$ // si stands for "smallest index"
3: **while** $P_{si}(\sigma) < n$ **do**
4: $S' \leftarrow \{J_j / P_j(\sigma) > P_{si}(\sigma)\}$
5: **while** $S' \neq \emptyset$ **do**
6: Let $J_k \in S'$ such that $k = argmax_{j \in S'}$ (biggest index first)
7: $\sigma' \leftarrow (\sigma_1 J_k) J_{[si]} \sigma_2 \sigma_3$ // remember that $\sigma = \sigma_1 J_{[si]} \sigma_2 J_k \sigma_3$
8: $S' \leftarrow S' \setminus \{J_k\}$
9: **if** σ' is feasible (respect of all the deadlines) **then** $\sigma \leftarrow \sigma'$
10: $si \leftarrow$ smallest index in S'

The complexity of FW1$_{index}$ is in $O(n^2)$.

Example 26 *Let consider the same data as in example 24. Before J_1 it is possible to sequence J_6. Then we have to sequence J_1. The new job with the smallest index is J_2, which has to be sequenced immediately. The new job with the smallest index is J_3. It is possible to put J_4 before J_3. Then, comes J_5.*

The final sequence is $(J_6, J_1, J_2, J_4, J_3, J_5)$ with a level equal to 9.

Example 27 *Consider the date of example 25. We consider the sequence $(J_1, J_2, ..., J_{10})$. It is possible to put before J_1 the jobs J_{10}, then J_9 and J_8, and J_5 to J_2. After J_1, the job with the smallest index is J_6 and we can put J_7 before it.*

The final sequence is $(J_{10}, J_9, J_8, J_5, J_4, J_3, J_2, J_1, J_7, J_6)$ with a level equal to 10.

Another Forward algorithm can be proposed.

Algorithm 17 FW2$_{index}$ is almost the same as Algorithm FW1$_{index}$, the only change is the step6 which becomes: Let $J_k \in S'$ such that $p_k = \min_{p_j \in S'} p_j$ (the smallest processing time first) and choose the biggest index first in case of ties.

The complexity of FW2$_{index}$ is $O(n^2)$.

Example 28 *Let consider the same data as in example 24. Before J_1 it is possible to sequence J_4. It is also possible to insert J_5, and (J_4, J_5) is resorted in (J_5, J_4). Then we have to sequence J_1, then J_2. Finally, J_3 and J_6.*

The final sequence is $(J_5, J_4, J_1, J_2, J_3, J_6)$ with a level equal to 8.

Algorithm 17 $FW2_{index}$ - $\sum N_j$

$\sigma = (J_1, J_2, ..., J_n)$
$si = 1$ // si stands for "smallest index"
while $P_{si}(\sigma) < n$ **do**
    $S' \leftarrow \{J_j / P_j(\sigma) > P_{si}(\sigma)\}$
    while $S' \neq \emptyset$ **do**
        Let $J_k \in S'$ such that $p_k = \min_{p_j \in S'} p_j$ (smallest processing time first)
        $\sigma' \leftarrow (\sigma_1 J_k) J_{[si]} \sigma_2 \sigma_3$ // remember that $\sigma = \sigma_1 J_{[si]} \sigma_2 J_k \sigma_3$
        $S' \leftarrow S' \setminus \{J_k\}$
        if σ' is feasible (respect of all the deadlines) **then** $\sigma \leftarrow \sigma'$
    $si \leftarrow$ smallest index in S'

3.3 Metaheuristic algorithms

We consider in the following the same "configuration" for all the metaheuristic algorithms. The algorithms described below are the generic algorithms for all the problems within the thesis. However, for each problem, we will present more details if there are necessary new parameters.

3.3.1 Common configuration

Initial solutions The initial solutions for all the metaheuristic algorithms is the best result of the heuristic methods BW_{index} and FW_{index}.

Terminating criterion The Terminating criterion of Metaheuristic methods is a computation time limit.

Diversification After intensifying exploring the accumulated search experience (by concentrating the search in a confined, small search space area), we may diversify the search by searching in other space to explore "in the large" of the search space. Intensification and Diversification are contrary and complementary.

To apply the diversification, we diversify the discovered space by a procedure changing the objective function after a limited iterations (it depends on the number of jobs).

Neighborhood operators There are many ways to define the neighborhood. We consider four types of neighbors called SWAP, EBSR, EFSR and Inversion.

We denote by S the current sequence: $S = S_1 S_{[i]} S_2 S_{[j]} S_3$, with S_1, S_2 and S_3 three subsequences of S and $S[i]$ and $S[j]$ the jobs in positions i and j ($i \neq j$) respectively. The neighborhood operators are the following (illustrated in Fig. 3.2):

- SWAP: A neighbor of S is created by interchanging the jobs in position i and j, leading to sequence $S' = S_1 S_{[j]} S_2 S_{[i]} S_3$.

- EBSR: (Extraction and Backward Shifted Re-insertion): A neighbor of S is created by extracting $S_{[j]}$ and re-inserting $S_{[j]}$ backward just before $S_{[i]}$, leading to sequence $S' = S_1 S_{[j]} S_{[i]} S_2 S_3$.
- EFSR (Extraction and Forward Shifted Re-insertion): A neighbor of S is created by extracting $S_{[i]}$ and re-inserting it forward immediately after $S_{[j]}$, leading to a sequence $S = S_1 S_2 S_{[j]} S_{[i]} S_3$.
- Inversion: A neighbor of S is created by inserting $S_{[j]} \overline{S_2} S_{[i]}$ between S_1 and S_3, where $\overline{S_2}$ is the inverse of sequence S_2.

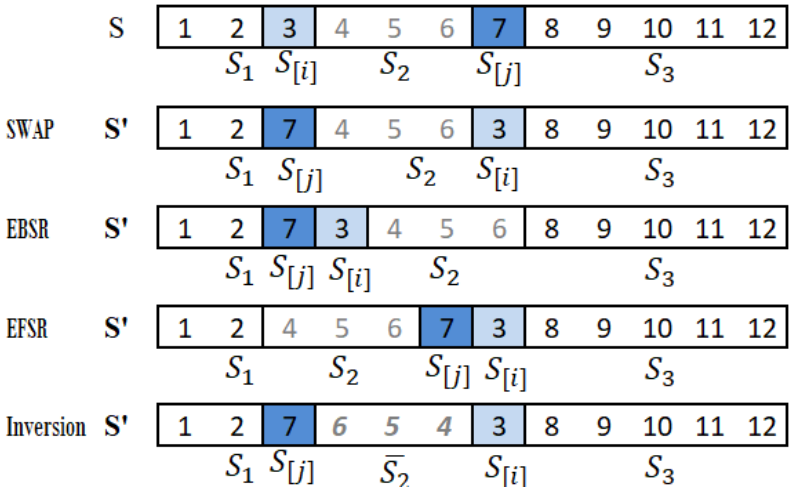

Figure 3.2: Neighnorhood

3.3.2 Algorithms

Tabu search In alg. 18 we denote by T the tabu list. BN indicates the best neighbor and fBN is the value of BN. The notation (X, k, j) with X ($X \in \{SWAP, EBSR, EFSR, Inversion\}$) indicates that the move X of k and j is Tabou. The notation $X(S, (k, j))$ returns a neighbor S' after move X.

Some additional elements have been implemented, and are not presented in the algorithm. For example, several flags have been introduced in order to check easily the influence of all the neighbors. Furthermore, the neighborhood sizes have been limited by some parameters in order to concentrate the search on the more promising parts of the search space.

Simulated Annealing The Simulated Annealing algorithm Alg.19 is the same as the one described in Alg. 2. The way to implement the neighborhood is the same as the one described in Alg. 18.

The value T_0 depends on the best value of polynomial time heuristic methods (BW, FW1, FW2) and the number of job n.

Algorithm 18 TS - $\sum N_j$

$S = S_0$, initial solution, the best among the solutions returned by BW, FW1 and FW2
$S^* = S_0$ // best solution of $N(S)$ and $non - tabu$
$f^* = f(S_0)$ // f^* best solution value
$T = \emptyset$ // T is the Tabu list.
while $CPU \leq TimeLimit$ **do**
    $fBN = \infty$
    for $k = 0$ **to** $n - 1$ **do**
        for $j = k + 1$ **to** n **do**
            if $(SWAP, k, j) \notin T$ **then**
                $S' = SWAP(S, (k, j))$, Calculate $f(S')$
                if $(f(S') < fBN)$ **then** $BN \leftarrow S', fBN \leftarrow f(S'), move = (k, j)$
            if $(EBSR, k, j) \notin T$ **then**
                $S' = EBSR(S, (k, j))$, Calculate $f(S')$
                if $(f(S') < fBN)$ **then** $BN \leftarrow S', fBN \leftarrow f(S'), move = (k, j)$
            if $(EFSR, k, j) \notin T$ **then**
                $S' = EFSR(S, (k, j))$, Calculate $f(S')$
                if $(f(S') < fBN)$ **then** $BN \leftarrow S', fBN \leftarrow f(S'), move = (k, j)$
            if $(Inversion, k, j) \notin T$ **then**
                $S' = EFSR(S, (k, j))$, Calculate $f(S')$
                if $(f(S') < fBN)$ **then** $BN \leftarrow S', fBN \leftarrow f(S'), move = (k, j)$
    $S \leftarrow BN$
    if $(fBN < f(S^*))$ **then** $S^* = BN, f(S^*) = fBN$
    $T \leftarrow T \cup \{move\}$
return (S^*)

Algorithm 19 SA - $\sum N_j$

$i \leftarrow 0$
$S = S_0$, initial solution, the best among the solutions returned by BW, FW1 and FW2
$T_0 =$ Initial temperature
while $CPU \leq TimeLimit$ **do**
    $S' \leftarrow$ Some random neighbor solution of S (of SWAP type)
    if $f(S') < best$ **then**
        $S^* \leftarrow S'$
        $best \leftarrow f(S')$
    else
        if $random[0, 1] < min\{1, e^{\frac{f(S)-f(S')}{T}}\}$ **then**
            $S := S'$
    $T_{i+1} \leftarrow g(T_i)$
    $i \leftarrow i + 1$
return (S^*)

3.4 Computational experiments

Some computational experiments have been conducted in order to evaluate the quality and the performances of the methods [Ta et Billaut, 2018a, Ta et Billaut, 2018b].

We detail the data generation, give the results and discuss the results.

3.4.1 Data generation

The computational experiments have been run on an Intel(5R) Core(TM) i5-63000 CPU 2.4 Go 2.5 GHz Memory RAM install: 16,0 Go (15,9 Go utilisable), using IBM ILOG CPLEX 12.6. Two types of instances have been generated. Instances of type I are classical random data sets, instances of type II are difficult random data sets. For each type, 30 instances have been generated for each value of n, with $n \in \{10, 20, ..., 100\}$.

For instances of type I, random data sets have been generated as follows:

- $p_j \in [1, 100]$, $w_j \in [1, 100]$
- $d_j \in [(\alpha - \beta/2)P, (\alpha + \beta/2)P]$ with $P = \sum p_j$, $\alpha = 0.75$ and $\beta = 0.25$

Then, these instances receive the pre-treatment which transforms the due dates into deadlines and renumbers the jobs in EDD order.

For instances of type II, random data sets have been generated as follows:

- for $n' = \lfloor n/4 \rfloor$ jobs:
  - $p_j = 1$, $w_j = 0$
  - $\tilde{d}_j = 4jP/n$ with P the sum of processing times of the remaining jobs
- for the $(n - n')$ remaining jobs:
  - $p_j \in [1, 100]$, $w_j = w_{0j} + P$, with $w_{0j} \in [1, 100]$ and $P = \sum_{j=1}^{n} p_j$
  - $\tilde{d}_j = P + \lfloor n/4 \rfloor$

These instances have been generated for the $1|\widetilde{d}_j|\sum w_j P_j$ problem (see Chapter 4). They do not need the pre-treatment because the first n' jobs are numbered in EDD order and the remaining jobs have the same deadline.

The CPU time to solve each instance has been limited to 180 seconds.

For setting the parameters of the tabu search and the simulated annealing algorithm, some preliminary tests have been performed. The results for their settings are presented in table 3.1.

	SA	TS
Classic Data	$T = int(BestH/n) + 1$ $a = 0.99 - n/1000$ $Limitedloop = 1000$	$Tabumax = 40 + int(2n/10)$ $Limitedloop = 1700 + n/10$
Difficult Data	$T = int(bestH/n) + 1$ $a = 0.99 - n/1000$ $Limitedloop = 300$	$Tabumax = 40 + int(2n/10)$ $Limitedloop = 1700 + n/10$

Table 3.1: Settings for SA and TS algorithms

3.4.2 Results

We present in this section the results of the computational experiments.

First, we denote some general notations that we will use for all the computational experiments of all our problems.

- Polynomial time heuristic methods: $H = \{BW, FW1, FW2\}$
- 'cpu' indicates the CPU time in seconds.
- 'opt' indicates the number of times the method found the optimal solution in less than 180 seconds.

Comparison of the performances of exact methods The results of the exponential methods MILP1, MILP2, MILP3 and $B\&B$ for instances of type I and II are presented in tables 3.2 and 3.3. MILP2 presenting bad performances, the experiments for classical instances have not been performed for n greater than 50. The DP algorithm has not been implemented.

$\sum N_j$	MILP1		MILP2		MILP3		B&B	
n	cpu	opt	cpu	opt	cpu	opt	cpu	opt
10	0.3	30	1.6	30	0.2	30	0.0	30
20	47.2	30	180	0	24.7	29	0.0	30
30	180	0	180	0	180	0	0.0	30
40	180	0	180	0	180	0	0.0	30
50	180	0	180	0	180	0	0.1	30
60	180	0	180	0	180	0	2.4	30
70	180	0	180	0	180	0	23.8	29
80	180	0	180	0	180	0	127.3	15
90	180	0	180	0	180	0	174.5	1
100	180	0	180	0	180	0	180.0	0

Table 3.2: Results of the exact methods for Type I instances

For type I instances, MILP2 can solve instances with up to 10 jobs while MILP1 and MILP3 can solve instances up to 20 jobs. MILP3 seems to be better than MILP1, but has the same limitations. The B&B algorithm is the best exact method, solving quite all instances with up to 70 jobs.

$\sum N_j$	MILP1		MILP2		MILP3		B&B	
n	cpu	opt	cpu	opt	cpu	opt	cpu	opt
10	0.7	30	0.11	30	0.05	30	0.0	30
20	152	10	111	20	0.7	30	0.5	30
30	180	0	180	0	30	30	180	0
40	180	0	180	0	180	0	180	0
50	180	0	180	0	180	0	180	0
60−100	180	0	180	0	180	0	180	0

Table 3.3: Results of the exact methods for Type II instances

For Type II instances, all the exact methods can solve instances with up to 20 jobs, except MILP1 and MILP2. MILP3 can solve all the instances with up to 40 jobs. For these instances, MILP3 is better than the B&B algorithm.

Comparisons of the quality of exact methods Except when they return the optimal solution, it is possible to compare the quality of the solution returned by the exact methods after within the allocated computation time of 180 seconds.

The results are presented in Tables 3.4 and 3.5. The columns indicate the number of times the method returns the best solution.

n	MILP1	MILP2	MILP3	B&B
10	30	30	30	30
20	30	8	30	30
30	1	0	7	30
40 - 100	0	0	0	30

Table 3.4: Comparison of the quality of exact methods for Type I instances

n	MILP1	MILP2	MILP3	B&B
10	30	30	30	30
20	23	29	30	30
30	0	0	30	30
40	0	0	29	30
50	0	0	2	30
60 - 100	0	0	0	30

Table 3.5: Comparison of the quality of exact methods for Type II instances

In terms of quality, it is clear that the B&B algorithm is always the best among the four exact methods. We note that MIP3 is very performing for the difficult instances.

Comparison of the polynomial time heuristic algorithms We consider now the polynomial time heuristic methods BW, FW1, FW2 (the computation time to solve an instance for all these methods is nearly zero).

The results for instances of type I is presented in Table 3.6.The results for instances of type II are presented in Table 3.7. Column # Best indicates the number of times the method is the best among all the heuristic methods, and Δ is the average deviation between the method and the best heuristic method.

$\sum N_j$	BW		FW1		FW2	
n	# Best	Δ	# Best	Δ	# Best	Δ
10	24	1.6%	27	0.6%	14	8.2%
20	10	11.6%	16	5.1%	18	4.0%
30	5	15.1%	11	10.3%	21	2.0%
40	3	17.1%	6	13.1%	27	0.2%
50	0	22.9%	3	17.2%	27	0.1%
60	0	23.3%	1	17.9%	29	0.2%
70	0	22.4%	2	18.1%	28	0.1%
80	0	25.6%	0	19.9%	30	0.0%
90	1	24.3%	0	18.5%	29	0.0%
100	0	28.5%	0	23.5%	30	0.0%

Table 3.6: Results of the polynomial heuristic methods for Type I instances

$\sum N_j$	BW		FW1		FW2	
n	# Best	Δ	# Best	Δ	# Best	Δ
10	29	0.3%	6	13.4%	2	29.7%
20	30	0.0%	0	26.1%	0	39.1%
30	30	0.0%	0	26.3%	0	36.6%
40	30	0.0%	0	28.8%	0	39.3%
50	30	0.0%	0	27.4%	0	37.0%
60	30	0.0%	0	28.5%	0	37.9%
70	30	0.0%	0	27.5%	0	37.0%
80	30	0.0%	0	28.1%	0	37.7%
90	30	0.0%	0	27.3%	0	37.4%
100	30	0.0%	0	27.2%	0	37.6%

Table 3.7: Results of the Polynomial heuristic methods for Type II instances

For Type I instances, the best method is clearly FW2 (except for $n = 10$ jobs). Heuristics BW and FW1 are on average at 19.2% and 14.4% respectively. However, for difficult instances, the behavior of the heuristics is not the same. BW becomes quite always the

best heuristic, and FW1 and FW2 are only at 26.1% and 36.9% of average deviation, respectively.

Comparison of the metaheuristics In this thesis, we consider the Tabu Seach (TS) method, and the Simulated Annealing (SA) method. These two metaheuristic methods use the same initial solution, which is the best result of the three polynomial time heuristic methods BW, FW1 and FW2. The parameters are given in Table 3.1. The results of the metaheuristic methods for instances of type I and II are presented in Tables 3.8 and 3.9. Column # Best indicates how many times the method is the best metaheuristic, and column Δ is the average deviation between the method and the best metaheuristic. Column Δ^H is the average relative deviation between the method M and the best polynomial time heuristic H:

$$\Delta^H = \frac{H - M}{H}$$

$\sum N_j$	TS			SA		
n	# Best	Δ	Δ^H	# Best	Δ	Δ^H
10	30	0.0%	0.4%	30	0.0%	0.4%
20	27	0.2%	3.9%	22	0.5%	3.6%
30	19	1.8%	4.2%	22	0.4%	5.5%
40	15	3.1%	4.2%	19	0.5%	6.8%
50	14	2.4%	3.8%	19	0.4%	5.8%
60	15	1.6%	4.8%	17	0.5%	5.9%
70	13	2.5%	4.1%	21	0.3%	6.3%
80	12	2.9%	4.4%	21	0.4%	6.7%
90	14	2.1%	3.2%	16	0.5%	4.8%
100	12	1.8%	1.8%	20	0.4%	3.1%

Table 3.8: Results of the Metaheuristic methods for Type I instances

	TS			SA		
n	# Best	Δ	Δ^H	# Best	Δ	Δ^H
10 −100	30	0%	0%	30	0%	0%

Table 3.9: Results of the Metaheuristic methods for Type II instances

For type I instances, the two methods return similar results, but SA is the best method with a total of 207 best solutions, and only 171 for TS. The methods improve the initial solution, on average 3.5% for the TS and 4.9% for the SA.

For type II instances, the two methods are not performing. The results indicate that the methods are not able to improve the initial solution within 180 seconds.

Comparison of the exact and the metaheuristic methods We compare now the quality of the solutions returned by the exact methods and the metaheuristics. The results

are given in Tables 3.10 and 3.11. Column # B indicates the number of best solutions and Δ is the average relative deviation to the best solution. All the methods have a computation time limited to 180 seconds.

$\sum N_j$	MILP1		MILP2		MILP3		B&B		TS		SA	
n	#B	Δ	#B	Δ	#B	Δ	#B	Δ	#B	Δ	#B	Δ
10	30	0.0%	30	0.0%	30	0.0%	30	0.0%	30	0.0%	30	0.0%
20	30	0.0%	8	6.7%	30	0%	30	0.0%	26	0.4%	20	0.6%
30	1	9.9%	0	40.8%	7	3.3%	30	0.0%	13	2.4%	11	1.0%
40	0	59.4%	0	77.7%	0	14.1%	30	0.0%	13	4.0%	7	1.6%
50	0	72.3%	0	79.9%	0	39.5%	30	0.0%	5	3.7%	0	1.7%
60	0	73.5%	0	80.5%	0	44.8%	30	0.0%	10	2.4%	0	1.3%
70	0	77.4%	0	82.1%	0	52.5%	29	0.3%	3	3.9%	1	1.7%
80	0	77.4%	0	82.6%	0	54.6%	20	1.2%	7	4.2%	7	1.7%
90	0	79.2%	0	82.6%	0	53.3%	4	4.0%	13	2.6%	13	1.0%
100	0	73.7%	0	82.2%	0	52.7%	1	3.4%	12	1.9%	19	0.5%

Table 3.10: Comparison of the Exact and Metaheuristic methods for Type I instances

$\sum N_j$	MILP1		MILP2		MILP3		B&B		TS		SA	
n	#B	Δ	#B	Δ	#B	Δ	#B	Δ	#B	Δ	#B	Δ
10	30	0%	30	0%	30	0%	30	0%	30	0%	30	0%
20	23	0.8%	29	0.1%	30	0%	30	0%	30	0%	30	0%
30	0	21.3%	0	37.4%	30	0%	30	0%	30	0%	30	0%
40	0	50.4%	0	76.2%	29	0%	30	0%	30	0%	30	0%
50	0	67.2%	0	75.1%	2	6%	30	0%	30	0%	30	0%
60	0	74.3%	0	76%	0	32.6%	30	0%	30	0%	30	0%
70	0	74.3%	0	75.2%	0	32.8%	30	0%	30	0%	30	0%
80	0	74.8%	0	75.5%	0	32.7%	30	0%	30	0%	30	0%
90	0	74.6%	0	75.1%	0	33.2%	30	0%	30	0%	30	0%
100	0	74.9%	0	75.4%	0	32.6%	30	0%	30	0%	30	0%

Table 3.11: Comparison of the Exact and Metaheuristic methods for Type II instances

For the instances of Type I, the B&B method is the best for instances with up to 80 jobs but for instances with 90 or 100 jobs, TS and SA are better. Again, one can see that SA is better than TS.

For type II instances, the B&B, TS and SA always return a solution with the same value. For TS and SA, it is clear because they do not improve BW algorithm. But the B&B is not initialized by the solution of BW, and nevertheless, the B&B cannot improve the solution of BW in 180 seconds.

In fact, for the difficult instances that have been generated, BW is a very performing method. Not always optimal because for one instance among 300, FW algorithms are better (see Table 3.7). This instance is the one which is used to illustrate BW and FW

algorithms in examples 25 and 27. But due to the quality of BW, TS and SA cannot improve the solution and the B&B has not the time to find a better solution.

3.5 Conclusion

This chapter deals with the most interesting and the most difficult problem $1|\widetilde{d}_j|\sum N_j$. This general problem remains open. We present resolution methods for finding a minimum sequence.

We first propose non-polynomial time methods: MILP models, one based on positional variables and one based on Linear Ordering variables, a Dynamic Programming formulation (DP) and a branch-and-bound algorithm (B&B). We also consider polynomial time greedy algorithm: a backward algorithm and two forward algorithms. We also propose some metaheuristic methods: a tabu search and a simulated annealing algorithm.

The computational results are presented and compared. Especially, for type I instance, Simulated Annealing clearly improves the results with big values of n.

In general, for Type I instances, B&B is the best methods for $n < 80$ and SA is the best methods with $n > 80$. For Type II instances, B&B is the best method for $n = 10$, and the polynomial time heuristic methods BW is the best method for $n > 10$. The superiority of BW makes sense in Type II instances and brings us the curiosity, that is also an important point for our future research, even it was not optimal for one instance.

Chapter 4

Minimization of objective functions based on jobs positions

We consider in this chapter some objective functions based on jobs positions. After solving some basic cases, we propose some complexity results and resolution methods for the minimization of the total weighted positions of jobs. Then, we consider the case where the weight is equal to the job index.

4.1 Introduction and first results

In Section 1.3.1, we proposed to consider some problems related to the positions of jobs. In this section, we present some results.

We denote by P_j the position of job J_j in the sequence, $\forall j \in \{1, 2, ..., n\}$.

With this indicator, we propose the following objective functions:

$$P_{\max} \text{ the maximum position of a job} \tag{4.1}$$

$$\sum P_j \text{ the total positions of jobs} \tag{4.2}$$

$$\sum w_j P_j \text{ the total weighted positions of jobs} \tag{4.3}$$

The new objective functions related to the jobs positions are very particular because these objective functions, as for $\sum N_j$, do not depend on the jobs completion times.

4.1.1 First results with common (or without) due date

We present in this section the first results when there is no deadline. Some of them are trivial.

- solving the $1|\beta|P_{\max}$ problem has no interest for any field β since $P_{\max} = n$ whatever the sequence is.
- solving the $1|\beta|\sum P_j$ problem has no interest for any field β since $\sum P_j$ is always equal to $n(n+1)/2$.

- solving the $1||\sum jP_j$ problem is very easy. Sequence $\sigma^{\perp} = (J_n, J_{n-1}, ..., J_1)$ is always optimal and the objective function value is equal to $\frac{n(n+1)(n+2)}{6}$.

Proposition 7 *The $1||\sum w_jP_j$ can be solved to optimality by sorting the jobs in their weight non increasing order.*

Proof. Suppose we have a sequence $S = \sigma_1 J_i J_j \sigma_2$ with σ_1 and σ_2 two subsequences and J_i and J_j two consecutive jobs. We denote by $S' = \sigma_1 J_j J_i \sigma_2$ and WP for $\sum w_jP_j$.

$WP(S) \leq WP(S')$
$\Leftrightarrow WP(\sigma_1) + w_iP_i(S) + w_jP_j(S) + WP(\sigma_2) \leq WP(\sigma_1) + w_jP_j(S') + w_iP_i(S') + WP(\sigma_2)$
$\Leftrightarrow w_iP_i(S) + w_j(P_i(S) + 1) \leq w_jP_i(S) + w_i(P_i(S) + 1)$
$\Leftrightarrow w_j \leq w_i$

This is true because the positions of the jobs in σ_1 and σ_2 are the same in S and in S'.

So there is always an interest to put first the job with the maximum weight. The problem can be solved in $O(n \log n)$.

Notice also that this problem is equivalent to the $1|p_j = 1|\sum w_jC_j$ because with unitary jobs, the completion time of a job is also its position. We obtain the WSPT rule (Section 1.1.5.1) with $p_j = 1$ for all the jobs.

If there is no specific constraint in the field β involving the processing times (such as r_j or $\widetilde{d}_j$), the indicator P_j is equal to the indicator C_j with the constraint $p_j = 1$. So, all the results for scheduling problems of type $\alpha|\beta, p_j = 1|\sum(w_j)C_j$ are valid for problems $\alpha|\beta|\sum(w_j)P_j$.

4.1.2 First results with deadlines

We present in this section the first results when we consider deadlines.

- solving the $1|\widetilde{d}_j|P_{\max}$ problem has no interest since $P_{\max} = n$ whatever the sequence is. Sequence $\sigma^{\top} = (J_1, J_2, ..., J_n)$, which is supposed to be feasible, is optimal.
- solving the $1|\widetilde{d}_j|\sum P_j$ problem has no interest since $\sum P_j$ is always equal to $n(n+1)/2$. Sequence $(J_1, J_2, ..., J_n)$ is optimal.
- solving the $1|p_j = 1, \widetilde{d}_j|\sum jP_j$ problem. This problem is equivalent to the problem $1|p_j = 1, \widetilde{d}_j|\sum jC_j$. We have the polynomial algorithm 8 BW_{index} to solve it with the complexity in $O(n \log n)$.
- solving the $1|p_j = 1, \widetilde{d}_j|\sum w_jP_j$ problem. This problem is equivalents with the problem $1|p_j = 1, \widetilde{d}_j|\sum w_jC_j$. We have the polynomial exact method BW_{weight} to solve this problem with the complexity in $O(n \log n)$ (see also [Chen et Bulfin, 1990]).

However, solving the $1|\widetilde{d}_j|\sum jP_j$ problem remains an open question.

An interesting problem is the $1|\widetilde{d}_j|\sum w_jP_j$. The objective function $\sum w_jP_j$ can be considered as an interesting special case of $\sum w_jC_j$, which we consider below.

4.2 Total Weighted Positions

We consider in this section the minimization of the total weighted positions, subject to deadlines. The main content of this section has been presented in IESM 2017 conference (see [Ta *et al.*, 2017]).

4.2.1 Complexity

The problem we are interested here is the $1|\widetilde{d}_j|\sum w_jP_j$. We know that problem $1|p_j = 1|Lex(T_{\max}, \sum w_jC_j)$ and therefore problem $1|p_j = 1, \widetilde{d}_j|\sum w_jC_j$ can be solved in polynomial time [Chen et Bulfin, 1990], but problem $1|\widetilde{d}_j|\sum w_jC_j$ is strongly NP-hard ([Lenstra *et al.*, 1977], reduction from Subset Sum problem).

Proposition 8 *Problem* $1|\widetilde{d}_j|\sum w_jP_j$ *is strongly NP-hard.*

Proof. It is clear that the problem is in NP. Checking a solution can be done in polynomial time.

Then, we proceed by reduction from 3-PARTITION problem (see Section 1.1.2.1).

From an arbitrary instance of 3-PARTITION, we build an instance to our problem as follows:

- We define $\overline{p} = |A| \times \max(A)$.
- We define m dummy jobs J_i, $1 \leq i \leq m$ such that $p_i = 1$, $w_i = 0$, $\widetilde{d}_i = i(B + 3\overline{p} + 1)$.
- We add $3m$ regular jobs $J_{m+j}, 1 \leq j \leq 3m$ with $p_{m+j} = w_{m+j} = a_j + \overline{p}$ and $\widetilde{d}_{m+j} = m(B + 3\overline{p} + 1)$.

We show (Part I) that if we have an optimal solution to our problem, then we have a 3-PARTITION and (Part II) that if we have a 3-PARTITION, then we have an optimal solution to our problem.

Part (I) We first show that in an optimal solution, each job J_i is precisely in position $4i$, $1 \leq i \leq m$ and then we show that it completes exactly at its deadline if and only if the answer to 3-PARTITION is yes.

(a) We consider an optimal sequence S^*. We prove by induction that in S^*, each job $J_i, 1 \leq i \leq m$, is at position $4i$.

• First, we have to check it for $i = 1$.

Suppose that J_1 is in position k and suppose that it completes at time $C_1 \leq \widetilde{d}_1$. We denote by D the set of the first $k-1$ jobs (before J_1). We denote by J_l the job in position $k+1$, and we know that this job completes after $\widetilde{d}_1$ (since otherwise swapping J_1 and J_l would lead to a better solution). We denote by Δ the difference between the starting time of J_1 and $\widetilde{d}_1$. These notations are illustrated in Fig. 4.1.

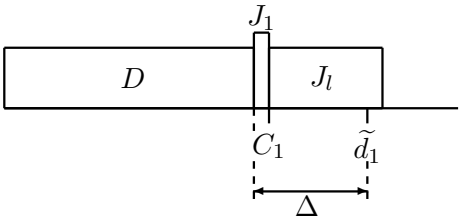

Figure 4.1: Notations for the position of job J_1

We have:

$$\begin{aligned}\Delta \geq 1 \Rightarrow \widetilde{d}_1 - \sum_{J_j \in D} p_j \geq 1 \\ \Rightarrow \widetilde{d}_1 - \sum_{J_j \in D} a_j - (k-1)\overline{p} \geq 1 \\ \Rightarrow B + 3\overline{p} + 1 - \sum_{J_j \in D} a_j - k\overline{p} + \overline{p} \geq 1 \\ \Rightarrow B + 4\overline{p} - \sum_{J_j \in D} a_j - k\overline{p} \geq 0\end{aligned}$$

So we have:

$$k \leq 4 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j) \tag{4.4}$$

Because $C_l > \widetilde{d}_1$, we have:

$$\begin{aligned}\sum_{J_j \in D} a_j + (k-1)\overline{p} + 1 + a_l + \overline{p} > B + 3\overline{p} + 1 \\ \sum_{J_j \in D} a_j + k\overline{p} + a_l > B + 3\overline{p}\end{aligned}$$

So we have:

$$k > 3 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j - a_l) \tag{4.5}$$

We deduce from (4.4), (4.5) that:

$$\begin{aligned}3 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j - a_l) < k \leq 4 + \frac{1}{\overline{p}}(B - \sum_{J_j \in D} a_j) \\ \Leftrightarrow 3 + \varepsilon_1 < k \leq 4 + \varepsilon_2\end{aligned}$$

It shows that J_1 is necessary in position 4.

• Suppose now that job J_i is in position $4i, \forall i, 1 \leq i \leq m-1$. We want to show that job J_{i+1} is in position $4(i+1)$. Suppose that J_{i+1} is in position k. We denote by D the

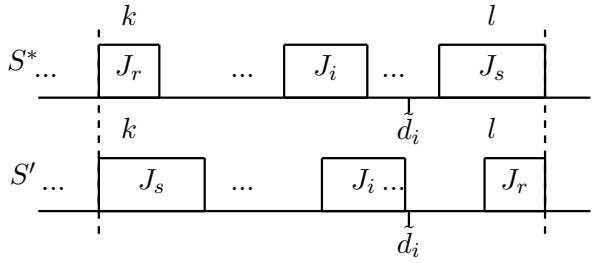

Figure 4.2: Case where J_i does not complete at time $\tilde{d}_i$ $(1 \leq i \leq m)$

jobs preceding J_{i+1}. There are $(k-1-i)$ jobs in D that are regular jobs. We have: $C_{i+1} \leq \widetilde{d}_{i+1}$
$\Rightarrow (i+1)(B+3\overline{p}+1) - (\sum_{J_j \in D} a_j + (k-1-i)\overline{p}) - i - 1 \geq 0$

$$\Rightarrow k \leq 4(i+1) + \varepsilon \tag{4.6}$$

Let us denote by J_l the job just after J_{i+1}. We have:
$C_l > \tilde{d}_{i+1} \Rightarrow \sum_{J_j \in D} a_j + |D|\overline{p} + i + 1 + \overline{p} + a_l > (i+1)(B+3\overline{p}+1)$
If J_{i+1} is in position k, the number of jobs before J_{i+1} is $k-1$, and before this job, there are $|D|$ regular jobs and i dummy jobs. So we have $k-1 = |D| + i$. Hence:

$$\sum_{J_j \in D} a_j + (k-1-i)\overline{p} + i + 1 + \overline{p} + a_l > (i+1)(B+3\overline{p}+1) \tag{4.7}$$

$$\Rightarrow k > (4i+3) + \varepsilon \tag{4.8}$$

We deduce from (4.6) and (4.8) that $k = 4i+4$.

(b) We show now that S^* is a 3-PARTITION, i.e. each dummy job completes exactly at its deadline.

We show that breaking the partition would either violate a deadline or generate a weaker solution.

Suppose that we do not follow the 3-PARTITION solution and that in S^*, we have $C_i < \widetilde{d}_i$ for one job J_i with $1 \leq i \leq m$ (i.e. we have an optimal solution that is not a 3-partition). Suppose there are two jobs: J_r before J_i in position k and J_s after J_i in position l such that if the jobs are swapped, then $C_i = \widetilde{d}_i$. In this case, illustrated in Fig. 4.2 we must have $a_r < a_s$ and $k < l$.

We denote by S' the solution S^* except that J_r and J_s are swapped in S'. We have:

$$\begin{aligned} Z(S') - Z(S^*) &= k(a_s + \overline{p}) + l(a_r + \overline{p}) - k(a_r + \overline{p}) - l(a_s + \overline{p}) \\ &= k(a_s - a_r) + l(a_r - a_s) \\ &= (l-k)(a_r - a_s) < 0 \end{aligned}$$

This quantity is negative, which means that S^* is not optimal, so this case is not possible.

Suppose now that there are two jobs J_{r_1} and J_{r_2} before J_i and two jobs J_{s_1} and J_{s_2} after J_i so that if the jobs are swapped, then $C_i = \widetilde{d}_i$. In this case, we have $a_{s_1} + a_{s_2} > a_{r_1} + a_{r_2}$ and :
$Z(S') - Z(S^*) = k_1 a_{s_1} + k_2 a_{s_2} + l_1 a_{r_1} + l_2 a_{r_2} - k_1 a_{r_1} + k_2 a_{r_2} + l_1 a_{s_1} + l_2 a_{s_2} = (k_1 - l_1)(a_{s_1} - a_{r_1}) + (k_2 - l_2)(a_{s_2} - a_{r_2})$

We know that $k_1 - l_1 < 0, k_2 - l_2 < 0$
$\Rightarrow \max(k_1 - l_1, k_2 - l_2) < 0$ and we have
$Z(S') - Z(S^*) = (k_1 - l_1)(a_{s_1} - a_{r_1}) + (k_2 - l_2)(a_{s_2} - a_{r_2})$
$< (a_{s_1} - a_{r_1}) \max(k_1 - l_1, k_2 - l_2) + (a_{s_2} - a_{r_2}) \max(k_1 - l_1, k_2 - l_2) < (a_{s_1} - a_{r_1} - a_{s_2} - a_{r_2})) \max(k_1 - l_1, k_2 - l_2) < 0$

For the same reasons, this case is not possible because S^* is optimal. By extending this reasoning to any number of jobs to exchange (same number before and after J_1 due to the position of this job), we prove that if the answer to the 3-PARTITION problem is YES, we must follow the 3-PARTITION solution and therefore we have a feasible solution that is optimal.

Part (II) Suppose there is an answer YES to the 3-PARTITION problem. In this case, we can take the 3-partitions of the jobs and put a dummy job between each 3-partition. Because of the deadlines definition, each dummy job completes exactly at its deadline. The contribution to the objective function of the jobs of a 3-partition can be determined. Let consider the three jobs of the 3-partition number i: J_a, J_b and J_c. Of course, these jobs are put in their 3-partition in their weight decreasing order. Suppose that $w_a > w_b > w_c$ and that J_a is in position k. The contribution of these three jobs is equal to $kw_a + (k+1)w_b + (k+2)w_c$. This quantity is equal to $k(w_a + w_b + w_c) + w_b + 2w_c$. Because $w_a + w_b + w_c = B + 3\overline{p}$, it is the same quantity for all the 3-partitions, we see that the contribution of J_a, J_b and J_c do not depend on the position k of J_a. So any sequence where the 3-partitions are put between the dummy jobs have the same objective function value and are optimal.

4.2.2 Properties and particular cases

4.2.2.1 Hardy's inequalities

G.H. Hardy has proposed some results in number theory and mathematical analysis, reported in [Hardy *et al.*, 1934]. Among his results, we present the following. Let consider two vectors of integers u and v of size n. Suppose that:

$$\begin{cases} 0 \leq u_1 \leq u_2 \leq ... \leq u_n \\ 0 \leq v_1 \leq v_2 \leq ... \leq v_n \end{cases} \tag{4.9}$$

Then, for any permutation σ, we have:

$$\sum_{j=1}^{n} u_j v_{n+1-j} \leq \sum_{j=1}^{n} u_j v_{\sigma(j)} \leq \sum_{j=1}^{n} u_j v_j \tag{4.10}$$

There is equality in 4.10 if and only if at least one of the three following conditions is satisfied:

- $u_1 = u_2 = ... = u_n$
- $v_1 = v_2 = ... = v_n$
- $v_j = v_{\sigma(j)}, \forall j$ for the right sign "=", $v_{(n+1-j)} = v_{\sigma(j)}, \forall j$ for the left sign "=".

We can deduce that if

$$\begin{cases} 0 \leq u_1 \leq u_2 \leq ... \leq u_n \\ v_1 \geq v_2 \geq ... \geq v_n \geq 0 \end{cases} \tag{4.11}$$

then for any permutation σ:

$$\sum_{j=1}^{n} u_j v_j \leq \sum_{j=1}^{n} u_j v_{\sigma(j)} \leq \sum_{j=1}^{n} u_j v_{n+1-j} \tag{4.12}$$

4.2.2.2 Particular case

Property 10 *Suppose that the weights are agreeable, i.e.* $w_1 \geq w_2 \geq ... \geq w_n$. *The problem is denoted by* $1|\widetilde{d}_j, agree|\sum w_j P_j$. *Then, sequence* $\sigma^\top$ *is the optimal sequence.*

Proof. We can apply Hardy's property.

$$\begin{cases} P_j : 0 \leq 1 \leq 2 \leq ... \leq n \\ w_j : w_1 \geq w_2 \geq ... \geq w_n \geq 0 \end{cases} \tag{4.13}$$

It satisfies 4.11, so we apply the inequality 4.12, we have

$$Z(\sigma^\top) = \sum_{j=1}^{n} j w_j \leq \sum_{j=1}^{n} j w_{\sigma(j)} \leq \sum_{j=1}^{n} j w_{n+1-j} \tag{4.14}$$

Evidently, $\sigma^\top$ is the optimal sequence.

4.2.3 Exact methods

4.2.3.1 MILP models

We propose two MILP models.

MILP1: Positional variables We use the position variables

$$x_{j,k} = \begin{cases} 1 & \text{if job } J_j \text{ is in position } k \\ 0 & \text{otherwise} \end{cases}$$

The expression of the position of job J_j is

$$P_j = \sum_{k=1}^{n} k x_{j,k}$$

Therefore, we have:

$$\text{MIN } \sum_{j=1}^{n} w_j P_j \tag{4.15}$$

$$\text{s.t. } (1.1), (1.2) \tag{4.16}$$

$$\sum_{q=1}^{k} \sum_{j=1}^{n} p_j x_{j,q} \leq \sum_{j=1}^{n} \widetilde{d}_j x_{j,k}, \forall k \in \{1, 2, ...n\} \tag{4.17}$$

$$P_j = \sum_{k=1}^{n} k x_{j,k}, \forall j \in \{1, 2, ...n\} \tag{4.18}$$

This model contains n^2 binary variables and n continuous variables.

MILP2: Linear Ordering variables We use the variables:

$y_{i,j} = \begin{cases} 1 & \text{if job } J_i \text{ precedes } J_j \\ 0 & \text{otherwise} \end{cases}$

The position of a job (see (2.6)) is given by

$$P_j = \sum_{i=1}^{n} y_{j,i}$$

We have the model:

$$\text{MIN } \sum_{j=1}^{n} w_j P_j \tag{4.19}$$

$$\text{s.t. } (3.16), (3.17), (3.18) \tag{4.20}$$

$$P_j = \sum_{i=1}^{n} z_{j,i} \tag{4.21}$$

4.2.3.2 Dynamic Programming

We can propose a Dynamic Programming algorithm for solving the problem.

Let us call $\mathcal{J}$ a group of jobs ending by J_j, $k = |\mathcal{J}|$ is the number of jobs in $\mathcal{J}$.

We define $G_j(\mathcal{J}, k) = \begin{cases} F_{j-1}(J \backslash \{J_j\}) + P_j w_k & \text{if } C_k \leq \widetilde{d}_k \\ 0 & \text{otherwise} \end{cases}$

We have the following recursive relation:

$$F_k(\mathcal{J}) = min_{j \in \mathcal{J}} \big(F_{k-1}(\mathcal{J} \backslash \{J_j\}) + K_j(\mathcal{J}, k) \big) \tag{4.22}$$

The initial condition is $F_0(\emptyset) = 0$ and we search for $F_n(\{J_1, J_2, ..., J_n\})$. Because k takes its values in $(1, 2, ..., n)$, this DP algorithm has an exponential time complexity.

Example 29 *We consider a 4-job example with* $p = (4, 6, 3, 5), w = (2, 4, 1, 3)$ *and* $\tilde{d} = (10, 15, 18, 18)$.

k = 0: $F_0(\emptyset) = 0$

k = 1: *all the jobs can be put in position 1.*

$\mathcal{J}$	$\{J_1\}$	$\{J_2\}$	$\{J_3\}$	$\{J_4\}$
j	1	2	3	4
Completion time C_j	4	6	3	5
$1.w_j$	2	4	1	3
$F_0(\mathcal{J}\backslash\{k\})$	0	0	0	0
$G_1(\mathcal{J}, k)$	2	4	1	3
$F_1(\mathcal{J}\backslash\{k\})$	2	4	1	3

k = 2: $G_2(\mathcal{J}, k) = F_1(\mathcal{J}\backslash\{k\}) + 2.w_k$

$F_2(\mathcal{J}\backslash\{k\}) = min_{k\in\mathcal{J}} G_2(\mathcal{J}, k)$

$\mathcal{J}$	$\{J_1, J_2\}$		$\{J_1, J_3\}$		$\{J_1, J_4\}$		$\{J_2, J_3\}$		$\{J_2, J_4\}$		$\{J_3, J_4\}$	
j	1	2	1	3	1	4	2	3	2	4	3	4
C_j	10		7		9		9		11		8	
$2.w_j$	4	8	4	2	4	6	8	2	8	6	2	6
$F_1(\mathcal{J} \setminus \{J_j\})$	4	2	1	2	3	2	1	4	3	4	3	1
$G_j(\mathcal{J}, 2)$	8	10	5	4	7	8	9	6	11	10	5	7
$F_2(\mathcal{J}\backslash\{J_j\})$	8		4		7		6		10		5	

k = 3: $G_3(\mathcal{J}, k) = F_2(\mathcal{J}\backslash\{k\}) + 3.w_k$

$F_3(\mathcal{J}\backslash\{k\}) = min_{k\in\mathcal{J}} G_3(\mathcal{J}, k)$

$\mathcal{J}$	$\{J_1, J_2, J_3\}$			$\{J_1, J_2, J_4\}$			$\{J_1, J_3, J_4\}$			$\{J_2, J_3, J_4\}$		
j	1	2	3	1	2	4	1	3	4	2	3	4
C_j		13			15			12			10	
$3.w_j$	∞	12	3	∞	12	9	∞	3	9	12	3	9
$F_2(\mathcal{J} \setminus \{J_j\})$	6	4	8	10	7	8	5	7	4	5	10	6
$G_j(\mathcal{J}, 3)$	∞	16	11	∞	19	17	∞	10	13	17	13	15
$F_3(\mathcal{J} \setminus \{J_j\})$		11			17			10			13	

k = 4: $G_4(\mathcal{J}, k) = F_3(\mathcal{J}\backslash\{k\}) + 4.w_k$

$F_4(\mathcal{J}\backslash\{k\}) = min_{k\in\mathcal{J}} G_4(\mathcal{J}, k)$

J	$\{J_1, J_2, J_3, J_4\}$			
j	*1*	*2*	*3*	*4*
C_j		*18*		
$4w_j$	∞	∞	*4*	*12*
$F_3(\mathcal{J} \setminus \{J_j\})$	*13*	*10*	*17*	*11*
$G_j(\mathcal{J}, 4)$	∞	∞	*21*	*23*
$F_4(\mathcal{J})$		*21*		

Result:

- *With* $\mathcal{J} = \{J_1, J_2, J_3, J_4\}$, $F_4(\mathcal{J}) = 21$ *corresponding to* $j = 3$. *Therefore,* J_3 *is at position 4.*
- *With* $\mathcal{J} = \{J_1, J_2, J_4\}$, $F_3(\mathcal{J}) = 17$ *corresponding to* $j = 4$, *therefore* J_4 *is at the position 3.*
- *With* $\mathcal{J} = \{J_1, J_2\}$, $F_2(\mathcal{J}) = 1$ *corresponding to* $j = 1$, *therefore* J_1 *is at position 2.*
- *Finally, with* $J = \{J_2\}$, *we have* J_2 *at position 1.*

If we have n jobs, we have to enumerate all parts of a set of size n, and there are 2^n parts of a set. So the complexity of this DP algorithm is in $O(2^n)$.

4.2.3.3 Branch-and-bound

The B&B method for problem $1|\widetilde{d}_j|\sum w_j P_j$ has the same characteristics as the B&B for the $1|\widetilde{d}_j|\sum N_j$:

- A set of unscheduled jobs S, sorted in the weight non-increasing order. If there are many jobs with the same weight, we put them in non-decreasing order of the processing time. And if they still have the same weight and processing time, we put them in decreasing order of due date.
- The **initial upper bound** is given by the best polynomial heuristic methods (Backward, Forward).
- The strategy of branching consists in adding a job of S at position $n - k$ in σ *respecting the deadlines.* Then a node is characterized by a partial ending sequence σ composed by k jobs (from position $n - k + 1$ to n). The job in S that is chosen to put in σ starts from the right of S.

The **Lower bound** is computed as follows: the partial sequence of jobs is completed by the unscheduled jobs (which are put at the beginning of the sequence) in their weight decreasing order, without deadline considerations.

While $S \neq \emptyset$, all the jobs that be chosen respect of their deadlines, so all the sequences are feasible. Then, we continue evaluation **lower bound** $LB(\sigma) = \sum w_j P_j, \forall j \in (S + \sigma)$.

We denote by N a node in the B&B algorithm. A node N is composed by a tuple (S, σ, lb, t) with S the unscheduled jobs, σ the current sequence, lb the lower bound of the node and t the starting time of σ. σ^* denotes the best current sequence.

$S = \{J_{a_1}, J_{a_2}, ..., J_{a_n}\}$ satisfies $w_{a_1} \geq w_{a_2} \geq ... \geq w_{a_n}$.

The algorithm Alg. 20 describes the branch-and-bound algorithm.

Algorithm 20 B&B - $\sum w_j P_j$

$S = \{J_{a_1}, J_{a_2}, ..., J_{a_n}\}$, $\sigma = \emptyset$, $lb = 0$, $t = \sum_{j=1}^{n} p_j$
$N = (S, \sigma, lb, t)$
π = sequence returned by a heuristic, $\sigma^* = \pi$, $UB = \sum w_j P_j(\pi)$
$Q \leftarrow N$
while $Q \neq \emptyset$ **do**
    $N = Q[1]$
    if $|\sigma(N)| = n$ **then**
        if $lb(N) < UB$ **then** Update UB and σ^*
    else
        for $J_{a_k} \in InverseS(N)$ **do**
            if $a_k = 1$ **then**
                $NewN \leftarrow (\emptyset, S(N) + \sigma(N), lb(N), 0)$, $Q \leftarrow NewN$
            else
                $NewN = \Big(S(N) \setminus \{J_{a_k}\}, (J_{a_k} + \sigma), lb\big(S(N) \setminus \{J_{a_k}\} + J_{a_k} + \sigma\big), t(N) - p_{a_k}\Big)$
                if $lb(NewN) < UB$ **then** $Q \leftarrow NewN$
return σ^*

4.2.4 Heuristic Methods

We propose in this section approximate algorithms.

4.2.4.1 Greedy algorithms

We denote by BW_{weight} the backward algorithm where we select at each iteration the feasible job with minimum weight.

Similarly, we denote by FW1_{weight} the forward algorithm where we try to put in the first position the jobs with maximum weight first. FW2_{weight} the forward algorithm where we try to put in the first position the jobs with smallest processing time first, and if there are the same processing time, put maximum weight first.

4.2.4.2 Metaheuristics

We change the objective function of the Tabu search and of the Simulated annealing algorithms. The initial solution is the best solution returned by the greedy algorithms.

4.3 Particular case with $w_j = j$

We have seen in Chapter 2 (function Z') that the level of a sequence was related to the weighted position of jobs, with the weight equal to the job index. We focus on this problem in this section.

4.3.1 Characteristics of $\sum jP_j$ objective function

Property 11 *$\sum jP_j$ is bounded by a polynomial function of n:*

$$\frac{1}{6}n(n+1)(n+2) \leq \sum jP_j \leq \frac{1}{6}n(n+1)(2n+1)$$

Proof. The best possible solution is always $\sigma^{\perp} = (J_n, J_{n-1}, ..., J_1)$ and the worse solution is always $\sigma^{\top} = (J_1, J_2, ..., J_n)$. Therefore, we have:

$$\begin{aligned}
\sum_{j=1}^{n} j(n-j+1) \leq \sum jP_j \leq \sum_{j=1}^{n} j^2 \\
n\frac{n(n+1)}{2} - \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \leq \sum jP_j \leq \frac{1}{6}n(n+1)(2n+1) \\
\frac{3n^2(n+1) - n(n+1)(2n+1) + 3n(n+1)}{6} \leq \sum jP_j \leq \frac{1}{6}n(n+1)(2n+1) \\
\frac{n(3n^2+3n-2n^2-3n-1+3n+3)}{6} \leq \sum jP_j \leq \frac{1}{6}n(n+1)(2n+1) \\
\frac{1}{6}n(n^2+3n+2) \leq \sum jP_j \leq \frac{1}{6}n(n+1)(2n+1) \\
\frac{1}{6}n(n+1)(n+2) \leq \sum jP_j \leq \frac{1}{6}n(n+1)(2n+1)
\end{aligned}$$

Corollary 2 *$\sum jP_j$ can take $\frac{1}{6}n(n^2-1)+1$ different values.*

Property 12 *Let us denote by Z^* the value of the optimal sequence σ^* and Z_H the value returned by any heuristic algorithm H. We have the ratio:*

$$\frac{Z_H}{Z^*} < 2 \tag{4.23}$$

Proof.

Let us denote by $Z^{\top}$ and $Z^{\perp}$ the values of $\sigma^{\top}$ and $\sigma^{\perp}$, respectively. The ratio between $Z^{\top}$ and $Z^{\perp}$ is equal to:

$$\begin{aligned}
\frac{Z^{\top}}{Z^{\perp}} &= \frac{\frac{1}{6}n(n+1)(2n+1)}{\frac{1}{6}n(n+1)(n+2)} \\
&= \frac{2n+1}{n+2} \\
&= 2 - \frac{3}{n+2} \\
&< 2
\end{aligned}$$

The ratio is valid between the worse and the best possible sequences, so the proposition is true.

Property 13 *Property 6 holds for the minimization of $\sum jP_j$. In other terms, there always exists an optimal solution where the jobs are gathered into batches of jobs in decreasing numbering order.*

Proof. The proof is similar to the one of Property 6. Suppose there is an optimal solution not satisfying this condition. Then, by a simple pairwise exchange argument, we can show that it is possible to find a better solution.

4.3.2 Particular cases

We consider in this section some particular cases.

4.3.2.1 Unit processing times

Suppose that the durations of jobs are all equal to 1. We consider problem $1|p_j = 1, \widetilde{d}_j|\sum jP_j$. This problem is equivalent to problem $1|p_j = 1, \widetilde{d}_j|\sum jC_j$, which can be solved in polynomial time by BW_{index} algorithm (see also [Chen et Bulfin, 1990]).

4.3.2.2 Fixed number of batches: $B = 1$

Suppose that the number of batches is fixed and equal to 1. Then, the case is the same as for problem $1|\widetilde{d}_j, B = 1|\sum N_j$ (see Section 2.3.5).

4.3.2.3 Fixed number of batches: $B = 2$

Suppose that the number of batches is fixed and equal to $B = 2$. The condition for having a feasible solution is the same as for objective function $\sum N_j$ (see Section 2.3.5): if $\sum_{J_j \in \mathcal{D}} p_j > \tilde{d}_1$ there is no feasible solution with only two batches, with $\mathcal{D} = \{J_j / \widetilde{d}_j < \sum_{j=1}^{n} p_j\}$.

Let suppose that $\sum_{J_j \in \mathcal{D}} p_j \leq \tilde{d}_1$. We have the case described in Fig. 2.14.

Last jobs in LPT order Consider the case where the jobs in $\mathcal{J} \setminus \mathcal{D}$ (which have the same deadline) are numbered in decreasing processing time order (LPT order). The Alg.13 is also applied for this problem, we only change the objective function to evaluate the solution.

Example 30 *We consider again the instance as in Example 21 with $n = 8$ jobs, processing times $p = (7, 5, 3, 9, 5, 4, 3, 2)$ and deadlines $\tilde{d} = (28, 29, 30, 38, 38, 38, 38, 38)$.*

The sequence for the problem $1|\widetilde{d}_j, B = 2, LPT|\sum jP_j$ and the problem $1|\widetilde{d}_j, B = 2, LPT|\sum N_j$ are the same. That is $(J_8, J_7, J_6, J_3, J_2, J_1, J_5, J_4)$ and only change the objective function $\sum jP_j = 135$.

Proposition 9 *Algorithm 13 GR is optimal for problem* $1|\widetilde{d}_j, B = 2, LPT|\sum jP_j$.

Proof. See Appendix 1.

4.3.3 Non-polynomial time algorithms

We take the same non-polynomial time methods as for the minimization of $\sum w_j P_j$: MILP1 with positional variables and MILP2 with linear ordering variables, by only changing the objective function definition. Notice that MILP2 here corresponds to MILP3 in Chapter 3.

The branch-and-bound is nearly the same as for $\sum N_j$. We only change the objective function $\sum jP_j$ to evaluate the lower bound.

4.3.4 Heuristic methods

We take the heuristic methods previously presented: Alg. 12 BWg_{index}, Alg. 16 FW1$_{index}$ and Alg. 17 FW2$_{index}$. These methods have been defined for $\sum N_j$ minimization, but are still valid for $\sum jP_j$.

4.3.5 Metaheuristic methods

We take the metaheuristic methods previously presented: TS and SA. With only need to update the objective function.

4.4 Computational experiment

The data which are used are the same as the ones described in Section 3.4.1.

The parameters for the TS and SA algorithms are given in Table 4.1.

$\sum w_j P_j$	SA	TS
Classic Data	$T = int(bestH/b\%) + 1$ $a = 0.99 - n/1000$ $Limitedloop = 1000$	$Tabumax = 40 + int(2 * n/10)$ $Limitedloop = 1700 + n/10$
Difficult Data	$T = int(bestH/b\%) + 1$ $a = 0.99 - n/1000$ $Limitedloop = 200$	$Tabumax = 40 + int(2 * n/10)$ $Limitedloop = 1700 + n/10$

Table 4.1: Settings for SA and TS algorithms

4.4.1 Computational experiments for $\sum w_j P_j$

We take the same notations as in Section 3.4.2 to present the computational results for the minimization of the total weighted positions.

Comparison of the performances of exact methods The results of the exponential methods MILP1, MILP2 and B&B for instances of type I and II are presented in Tables 4.2 and 4.3. The DP algorithm has not been implemented.

For type I instances, Tables 4.2 shows that MILP2 can solve instances with up to about more than 20 jobs, while B&B algorithm solves quite all instances with up to 70 jobs. Even if we can see that the performances of MILP1 and B&B are in the same range, MILP1 is the best exact method, solving quite all instances with up to 70 jobs.

$\sum w_j P_j$	MILP1		MILP2		B&B	
n	cpu	opt	cpu	opt	cpu	opt
10	0	30	0	30	0	30
20	0.1	30	3.5	30	0	30
30	0.3	30	99.6	14	0	30
40	0.9	30	139.6	10	0.1	30
50	2.5	30	157.6	5	0.5	30
60	5.1	30	171.8	3	0.8	30
70	23.4	30	180.0	0	11.9	29
80	45.9	28	179.7	1	53.1	23
90	78.4	22	180.0	0	87.2	18
100	127.5	13	180.0	0	139.2	11

Table 4.2: Results of the exact methods for Type I instances

For instances of type II, Table 4.3 shows that MILP1 and MILP2 have more optimal solutions than the B&B algorithm. MILP1 can solve instances with up to 40 jobs but the B&B can solve only instances with 10 jobs and MILP2 up to 30 jobs.

$\sum w_j P_j$	MILP1		MILP2		B&B	
n	cpu	opt	cpu	opt	cpu	opt
10	0.2	30	0.1	30	0.1	30
20	1.2	30	1.6	30	178.0	1
30	16.4	30	91.9	23	180.0	0
40	105.1	21	174.0	2	180.0	0
50−90	180.0	0	180.0	0	180.0	0

Table 4.3: Results of the exact methods for Type II instances

Comparisons of the quality of exact methods Again, we compare the quality of the solution returned by the exact methods, after the allocated computation time of 180 seconds.

The results are presented in Tables 4.4 and Tables 4.5. The columns $\#B(E)$ indicate the number of times the method returns the best solution between the exact methods, and $\Delta(E) = \frac{method - BestExact}{method}$ is the relative deviation between an exact method and the best exact method.

$\sum w_j P_j$	MILP1		MILP2		B&B	
n	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$
10	30	0%	30	0%	30	0%
20	30	0%	30	0%	30	0%
30	30	0%	30	0%	30	0%
40	30	0%	28	0.02%	30	0%
50	30	0%	20	0.05%	30	0%
60	30	0%	16	0.41%	30	0%
70	30	0%	5	0.90%	30	0%
80	30	0%	6	1.37%	28	0%
90	29	0%	7	1.03%	26	0.07%
100	24	0,01%	7	1.78%	23	0.02%

Table 4.4: Comparison of the quality of exact methods for Type I instances

For type I instances, we can see that despite the small number of instances solved to optimality, the B&B algorithm quite always finds the optimal solution. The method has not the time to prove the optimality, but the obtained solution is quite always optimal. MILP2, which is better than B&B in terms of number of instances solved to optimality, only find optimal solutions for up to 40 jobs. For more jobs, the method does not often return the best solution, but on average a solution at less than 2% of the best solution.

$\sum w_j P_j$	MILP1		MILP2		B&B	
n	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$
10	30	0%	30	0%	30	0%
20	30	0%	30	0%	2	0.30%
30	30	0%	25	0%	0	0.19%
40	30	0%	5	0.02%	0	0.10%
50	22	0.03%	8	0.06%	0	0.06%
60	2	0.32%	25	0.50%	3	0.01%
70	0	1.56%	6	10.27%	24	0%
80	0	3.61%	0	15.08%	30	0%
90	0	6.72%	0	15.93%	30	0.07%

Table 4.5: Comparison of the quality of exact methods for Type II instances

For type II instances, the quality of exact methods in Tables 4.5 show a more complicated behavior for $n \neq 10$.

- for $n \leq 50$, MILP1 is the best method, B&B is the worst method.
- for $n = 60$, MILP2 is the best method.
- for $n \geq 70$, B&B is the best method.

Moreover, the relative deviations $\Delta(E)$ are very small for all $n \leq 60$. In terms of

quality, it is clear that the B&B algorithm is always the best among all the exact methods when $n \geq 60$, with a very small value $\Delta(E)$, generally smaller than 0.07%.

The Fig.4.3 illustrates clearly the quality of exact methods for Type II instances.

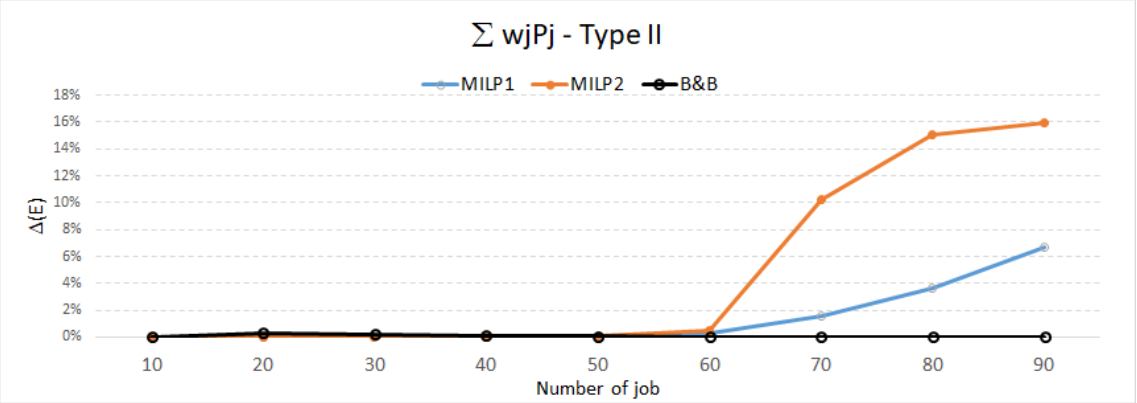

Figure 4.3: Graph of comparison of the exact methods for Type II instances

Comparison of the polynomial time algorithms We consider now the polynomial time heuristic methods BW, FW1 and FW2. Tables 4.6 and 4.7 give the results for instance of type I and II, respectively. Column $\#Best$ indicates the number of times the method is the best among all the heuristic methods, and $\Delta = \frac{method - BestH}{method}$ is the average deviation between the method and the best heuristic method. Column $\#opt$ indicates the number of times the heuristic returns the optimal solution (equal to the optimal solution returned by an exact method, if known).

$\sum w_j P_j$	BW			FW1			FW2		
n	$\#Best$	$\#opt$	Δ	$\#Best$	$\#opt$	Δ	$\#Best$	$\#opt$	Δ
10	28	28	0.1%	24	24	0.5%	27	27	0.07%
20	24	23	0.1%	19	19	1.1%	28	27	0.02%
30	18	18	0.2%	6	6	1.9%	26	24	0.02%
40	10	6	0.4%	9	6	1.4%	24	16	0.22%
50	10	3	0.1%	2	1	2.2%	24	11	0.01%
60	13	6	0.1%	1	0	1.6%	21	8	0.01%
70	9	3	0.2%	0	0	2.3%	23	4	0.01%
80	12	3	0.1%	2	0	1.9%	19	2	0.03%
90	10	3	0.2%	1	1	2.8%	20	3	0.01%
100	6	0	0.2%	0	0	1.5%	24	4	0.01%

Table 4.6: Results of the polynomial heuristic methods for Type I instances

For Type I instances, the best method is clearly FW2, even when $n = 10$ jobs, the number of best of BW algorithm is bigger than FW1 with a smaller average deviation. The relative deviation between the three methods is small, at most 2.8% for FW1, at most 0.4% for BW and at most 0.22% for FW2.

$\sum w_j P_j$	BW		FW1		FW2	
n	$\#Best$	Δ	$\#Best$	Δ	$\#Best$	Δ
10	0	7.3%	26	0.1%	4	1.9%
20	0	10.3%	30	0%	0	6.2%
30	0	13.1%	30	0%	0	7.5%
40	0	12.9%	30	0%	0	8.9%
50	0	13.5%	30	0%	0	9.2%
60	0	13.7%	30	0%	0	10.0%
70	0	14.2%	30	0%	0	10.3%
80	0	14.5%	30	0%	0	10.9%
90	0	15.1%	30	0%	0	10.9%

Table 4.7: Results of the polynomial heuristic methods for Type II instances

For type II instances, the behavior of the heuristics is not the same. FW1 becomes quite always the best heuristic with the average deviation nearly zero for all cases (except only when $n = 10$, with the very small value 0.1%), and BW and FW2 are only at around 13% and 10%, respectively. Notice that for $n = 10$, FW1 returns the optimal solution for one instance, and FW1 does not return any other optimal solution for $n \geq 20$.

Comparison of the metaheuristics The Tabu Search (TS) method and the Simulated Annealing (SA) method use the same initial solution, which is the best result of the three polynomial time heuristic methods BW, FW1 and FW2.

The results of the metaheuristic methods for instances of type I are presented in Table 4.8. Column $\#Best$ indicates how many times the method is the best metaheuristic, and column Δ is the average deviation between the method and the best metaheuristic. Column $\Delta^H = \frac{H-M}{H}$ is the average relative deviation between the method M and the best polynomial time heuristic H. Column $\#Imp$ indicates the number of times the metaheuristic improves the initial solution.

For type I instances, the relative deviation between the method and the best heuristic does not exceed 0.1%. The reason is that the value of the objective function is big (around 2000 for $n = 10$ jobs and around 200000 for $n = 100$ jobs). Column $\#Imp$ shows that the methods do improve the initial solutions except for $n \leq 30$, where these methods are quite always optimal.

The results of the metaheuristic methods for instances of type II are presented in Table 4.9. For type II instances, we can see that SA really improves the best heuristic (FW1) with a relative deviation around 10%, but TS is not really able to improve it. SA is the best metaheuristic method for this problem. Notice that the Tabu Search and the Simulated Annealing, except for one instance for $n = 10$, always improve the initial solution.

$\sum w_j P_j$	TS				SA			
n	$\#Best$	Δ	Δ^H	$\#Imp$	$\#Best$	Δ	Δ^H	$\#Imp$
10	30	0.00%	0.00%	0	30	0.00%	0.00%	0
20	29	0.01%	0.00%	0	30	0.00%	0.01%	1
30	29	0.00%	0.00%	0	30	0.00%	0.00%	1
40	26	0.01%	0.01%	8	30	0.00%	0.02%	12
50	19	0.09%	0.02%	12	29	0.00%	0.10%	18
60	12	0.06%	0.00%	0	30	0.00%	0.06%	18
70	10	0.04%	0.00%	0	30	0.00%	0.04%	20
80	20	0.05%	0.02%	18	19	0.00%	0.07%	15
90	15	0.03%	0.01%	15	24	0.00%	0.05%	22
100	17	0.03%	0.02%	20	19	0.00%	0.05%	20

Table 4.8: Results of the Metaheuristic methods for Type I instances

$\sum w_j P_j$	TS			SA		
n	$\#Best$	Δ	Δ^H	$\#Best$	Δ	Δ^H
10	30	0%	1.26%	30	0%	10.1%
20	15	0.009%	0.38%	30	0%	9.4%
30	6	0.014%	0.18%	29	0.0004%	10.2%
40	3	0.004%	0.10%	27	0.0001%	9.3%
50	3	0.004%	0.07%	29	0.0001%	10.4%
60	3	0.002%	0.04%	27	0.0002%	10.0%
70	1	0.001%	0.03%	29	0.0000%	10.4%
80	6	0.001%	0.03%	24	0.0001%	10.0%
90	0	0.001%	0.02%	30	0%	10.2%

Table 4.9: Results of the Metaheuristic methods for Type II instances

Comparison of exact and metaheuristic methods We compare now the quality of the solutions returned by the exact methods and the metaheuristics. The results are given in Tables 4.10 and 4.11. Column $\#B$ indicates the number of best solutions and Δ is the average relative deviation to the best solution of all the methods. All the methods have a computation time limited to 180 seconds.

$\sum w_jP_j$	MILP1		MILP2		B&B		TS		SA	
n	#B	Δ	#B	Δ	#B	Δ	#B	Δ	#B	Δ
10	30	0%	30	0%	30	0%	30	0%	30	0%
20	30	0%	30	0%	30	0%	29	0.006%	30	0%
30	30	0%	30	0%	30	0%	28	0.006%	29	0.002%
40	30	0%	28	0.015%	30	0%	26	0.013%	30	0%
50	30	0%	20	0,049%	30	0%	17	0.092%	27	0.005%
60	30	0%	16	0.406%	30	0%	12	0.068%	22	0.012%
70	30	0%	5	0,898%	30	0%	5	0.09%	10	0.049%
80	30	0%	6	1,367%	28	0.002%	9	0.14%	6	0.095%
90	29	0%	7	1.025%	26	0.066%	13	0.126%	9	0.092%
100	23	0.011%	7	1.779%	22	0.018%	11	0.066%	7	0.036%

Table 4.10: Comparison of the Exact and Metaheuristic methods for Type I instances

The performances of the exact methods MILP1 and B&B were very good for type I instances, and SA and TS have difficulties to be competitive for these instances.

$\sum w_jP_j$	MILP1		MILP2		B&B		TS		SA	
n	$\#B$	Δ	$\#B$	Δ	$\#B$	Δ	$\#B$	Δ	$\#B$	Δ
10	30	0%	30	0%	30	0%	30	0%	30	0%
20	30	0%	30	0.85%	2	0.39%	15	0.092%	30	0%
30	30	0%	25	6.74%	0	0.19%	5	0.014%	23	0.0006%
40	29	0.0%	5	8.36%	0	0.10%	1	0.005%	8	0.0010%
50	7	0.04%	0	9.45%	0	0.07%	1	0.004%	24	0.0003%
60	0	0.35%	0	9.5%	0	0.05%	3	0.002%	27	0.0002%
70	0	1.59%	0	9.99%	0	0.03%	1	0.001%	29	0.0000%
80	0	3.64%	0	9.64%	0	0.03%	6	0.001%	24	0.0001%
90	0	6.74%	0	9.86%	0	0.02%	0	0.001%	30	0%

Table 4.11: Comparison of the Exact and Metaheuristic methods for Type II instances

MILP1 is clearly the best method for $n \leq 40$. For $n \geq 50$, SA is the best method (show by the value $\Delta = 0$).

4.4.2 Computational experiments for $\sum jP_j$

We keep the same way to present the results as before.

Comparison of the performances of exact methods Consider first the exact methods for instance of type I in Table 4.12. From 10 to 50 jobs, $B\&B$ is better than MILP1 because the running times are smaller than for MILP1, but MILP1 is better than $B\&B$ when the number of jobs is greater than 50. In this case, MILP1 dominates all other

methods. Moreover, MILP1 has time enough to solve optimally almost all the instances in less than one minute, except one instance for $n = 100$ jobs.

$\sum jP_j$	MILP1		MILP2		B&B	
n	cpu	opt	cpu	opt	cpu	opt
10	0.0	30	0.0	30	0.0	30
20	0.1	30	87.1	21	0.0	30
30	0.4	30	99.6	0	0.1	30
40	1.2	30	180.0	0	0.3	30
50	2.4	30	180.0	0	1.3	30
60	5.9	30	180.0	0	13.3	30
70	8.9	30	180.0	0	57.4	29
80	14.5	30	180.0	0	150.0	9
90	25.6	30	180.0	0	175.2	1
100	52.7	29	180.0	0	180.0	0

Table 4.12: Results of the exact methods for Type I instances

However, for type II instances (Table 4.13), the exact methods are less performing. MILP1 and MILP2 can solve instances with up to 40 jobs, and B&B is the worst method, able to solve only the instances for $n = 10$ jobs.

$\sum jP_j$	MILP1		MILP2		B&B	
n	cpu	opt	cpu	opt	cpu	opt
10	0.1	30	0.0	30	0.0	30
20	1.2	30	0.3	30	180.0	0
30	12.9	30	6.8	30	180.0	0
40	103.0	26	93.4	23	180.0	0
50−100	180.0	0	180.0	0	180.0	0

Table 4.13: Results of the exact methods for Type II instances

Comparison of the quality of exact methods The results are presented in Tables 4.14 and 4.15. As for objective function $\sum w_j P_j$, the columns $\#B(E)$ indicate the number of times the method returns the best solution between the exact methods and $\Delta(E) = \frac{method - BestExact}{method}$ is the relative deviations between an exact method with the best of the exact methods.

For type I instances, the quality of exact methods in Tables 4.14 shows that the performances of MILP1, MILP2 and B&B is coherent with the performances of the exact methods presented in Table 4.12. The relative deviations $\#B(E)$ of B&B are smaller than 1.7%. However, for MILP2, the relative deviations $\#B(E)$ is up to 22.4%. So again, MILP1 dominates all the other exact methods.

$\sum jP_j$	MILP1		MILP2		B&B	
n	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$
10	30	0%	30	0%	30	0%
20	30	0%	29	0%	30	0%
30	30	0%	7	0.6%	30	0%
40	30	0%	2	2.7%	30	0%
50	30	0%	0	8.0%	30	0%
60	30	0%	0	15.2%	30	0%
70	30	0%	0	19.5%	29	0.1%
80	30	0%	0	21.6%	14	0.5%
90	30	0%	0	21.9%	4	1.4%
100	30	0%	0	22.4%	0	1.7%

Table 4.14: Comparison of the quality of exact methods for Type I instances

For type II instances, the quality of exact methods (see Table 4.15 and Fig. 4.4) shows that except for $n = 10$, the B&B method is not efficient. MILP1 and MILP2 have similar characteristics for $n \leq 40$. MILP1 is better for $n = 50$, but for $n \geq 60$, MILP2 is the most performing method with a relative deviation smaller than 0.32%.

$\sum jP_j$	MILP1		MILP2		B&B	
n	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$	$\#B(E)$	$\Delta(E)$
10	30	0%	30	0%	30	0%
20	30	0%	30	0%	0	1.48%
30	30	0%	30	0.6%	0	1.60%
40	29	0.00%	29	0.00%	0	1.63%
50	20	0.02%	13	0.04%	0	1.59%
60	9	0.11%	21	0.12%	0	1.54%
70	15	0.25%	15	0.32%	0	1.25%
80	10	0.57%	20	0.13%	0	0.57%
90	0	2.66%	30	0%	0	0.35%
100	0	5.27%	30	0%	0	0.27%

Table 4.15: Comparison of the quality of exact methods for Type II instances

Notice that the relative deviations $\#B(E)$ of B&B is very small, even when it does not return an optimal solution. The relative deviation is smaller than 1.63%.

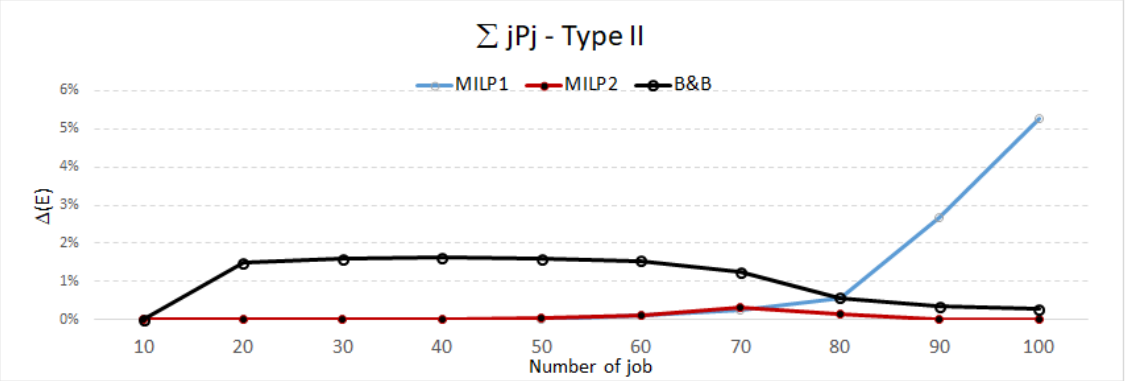

Figure 4.4: Graph of comparison of the exact methods for Type II instances

Comparison of the polynomial time heuristic algorithms The results for instances of type I are presented in Table 4.16. The results for instances of type II are presented in Table 4.17. Column $\#Best$ indicates the number of times the method is the best among all the heuristic methods, and $\Delta = \frac{method - BestH}{method}$ is the average deviation between the method and the best heuristic method.

$\sum jP_j$	BW		FW1		FW2	
n	$\#Best$	Δ	$\#Best$	Δ	$\#Best$	Δ
10	27	0.3%	30	0%	8	3.0%
20	12	1.8%	22	0.4%	8	2.1%
30	8	2.4%	16	1.3%	14	1.2%
40	9	2.6%	15	1.7%	15	0.9%
50	1	3.8%	10	2.3%	20	0.6%
60	2	3.5%	7	2.3%	23	0.4%
70	2	3.1%	7	2.1%	23	0.2%
80	2	3.7%	5	2.4%	24	0.3%
90	2	3.4%	8	1.9%	22	0.3%
100	1	4.4%	2	3.0%	28	0.1%

Table 4.16: Results of the polynomial heuristic methods for Type I instances

For Type I instances, FW1 is the best method and FW2 is the worst for $10 \leq n \leq 20$. But for all $n \geq 30$, BW is the worst while FW2 is the best method. However, the relative deviation values are not important, smaller than 4.4% for BW, and smaller than 3.0% for FW1 and FW2.

Fig. 4.5 illustrates more clearly the characteristics of three methods.

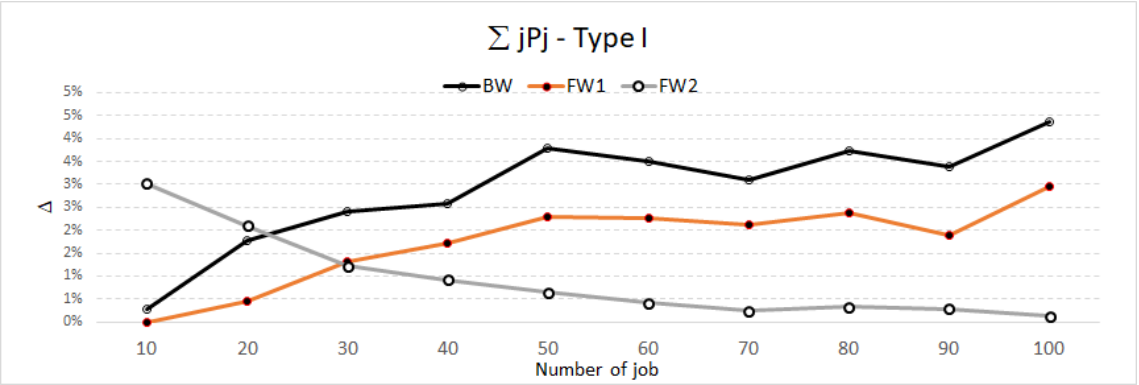

Figure 4.5: Graph of the heuristic methods for Type I instances

For Type II instances, the behavior of the polynomial heuristics is very easy to analyse, but it is not the same as before.

$\sum jP_j$	BW		FW1		FW2	
n	$\#Best$	Δ	$\#Best$	Δ	$\#Best$	Δ
10	29	0.01%	3	3.9%	1	9.9%
20	30	0%	1	7.4%	0	13.7%
30	30	0%	0	9.3%	0	15.6%
40	30	0%	0	9.8%	0	16.5%
50	30	0%	0	9.8%	0	17.0%
60	30	0%	0	10.1%	0	17.0%
70	30	0%	0	9.4%	0	17.3%
80	30	0%	0	9.1%	0	17.6%
90	30	0%	0	8.9%	0	18.1%
100	30	0%	0	9.1%	0	18.2%

Table 4.17: Results of the polynomial heuristic methods for Type II instances

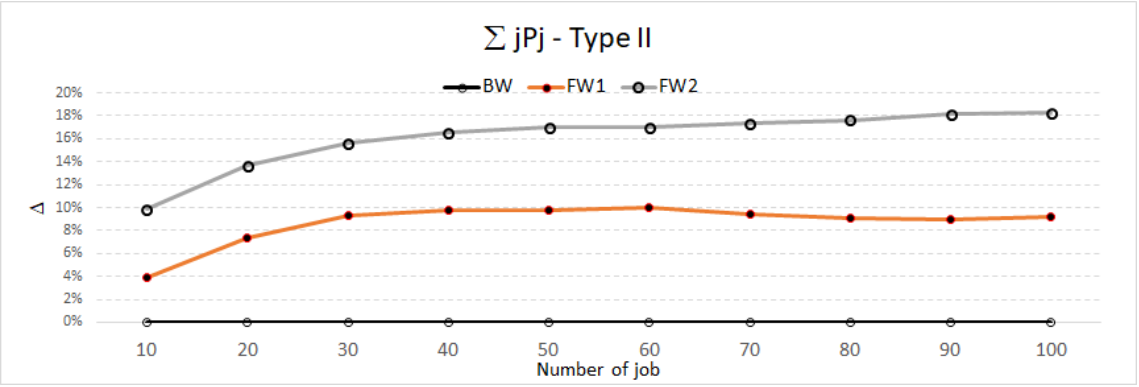

Figure 4.6: Graph of the heuristic methods for Type II instances

Tables 4.17 and Fig.4.6 show clearly that FW2 is the worse method and BW is the best for all value n. The relative deviations of FW1 is smaller than 10.1%, and of FW2 is smaller than 18.2%.

Comparison of the metaheuristics The parameters are given in Table 4.18.

$\sum jP_j$	SA	TS
Difficult Data	$T = int(bestH/(10 * n))$	$Tabumax = 40 + int(2 * n/10)$
	$a = 0.99 - n/1000$	
	$Limitedloop = 150$	$Limitedloop = 1700 + n/10$

Table 4.18: Settings for SA and TS algorithms

Because the exact methods are optimal for quite all the instances of type I, the experiments have been done for type II instances only. The results of the metaheuristic methods for instances of type II are presented in Table 4.19. Remember that the column $\#Best$ indicates how many times the method is the best metaheuristic, and column Δ is the average deviation between the method and the best metaheuristic. Column $\Delta^H = \frac{H-M}{H}$ is the average relative deviation between the method M and the best polynomial time heuristic H.

$\sum jP_j$	TS			SA		
n	$\#Best$	Δ	Δ^H	$\#Best$	Δ	Δ^H
10	30	0.00%	1.5%	30	0.00%	1.5%
20	20	0.10%	1.4%	30	0.00%	1.5%
30	2	0.50%	1.1%	29	0.00%	1.6%
40	1	0.40%	1.3%	29	0.00%	1.6%
50	4	0.50%	1.1%	26	0.01%	1.5%
60	0	0.40%	1.2%	30	0.00%	1.5%
70	0	0.40%	1.1%	30	0.00%	1.4%
80	1	0.40%	1.1%	29	0.00%	1.4%
90	0	0.50%	0.9%	30	0.00%	1.4%
100	0	0.50%	0.8%	30	0.00%	1.3%

Table 4.19: Results of the Metaheuristic methods for Type II instances

For type II instances, the two methods are executed, the results indicate that the methods are able to improve the initial solution within 180 seconds. The relative deviation Δ between TS and SA are not so far, smaller than 1.5% for TS and nearly zero for all n (except when $n = 50$) for SA. Especially, SA is the best method for all the values of n, and SA can improve up to $\Delta^H = 1.6\%$ the best result of the polynomial time heuristic methods.

Comparison of the exact and the metaheuristic methods We compare now the quality of the solutions returned by the exact methods and the metaheuristics. Since there is no metaheuristic method TS and SA considered for type I instances, we have only the comparison between the exact and the metaheuristic methods of type II instance. The results are given in Table 4.20. Column $\#B$ indicates the number of best solutions and Δ is the average relative deviation to the best solution. All the methods have a computation time limited to 180 seconds.

$\sum jP_j$	MILP1		MILP2		B&B		TS		SA	
n	$\#B$	Δ	$\#B$	Δ	$\#B$	Δ	$\#B$	Δ	$\#B$	Δ
10	30	0%	30	0%	30	0%	30	0%	30	0%
20	30	0%	30	0%	0	1.5%	20	0.1%	30	0%
30	30	0%	30	0%	0	1.6%	2	0.5%	24	0.0%
40	29	0.0%	29	0.0%	0	1.6%	0	0.4%	9	0.0%
50	20	0.0%	11	0.0%	0	1.6%	1	0.5%	4	0.1%
60	4	0.1%	14	0.2%	0	1.6%	0	0.4%	12	0.0%
70	1	0.5%	7	0.5%	0	1.5%	0	0.4%	22	0.0%
80	0	1.4%	2	1.0%	0	1.4%	1	0.4%	27	0.0%
90	0	3.7%	0	1.1%	0	1.4%	0	0.5%	30	0%
100	0	6.2%	0	1.0%	0	1.3%	0	0.5%	30	0%

Table 4.20: Comparison of the Exact and Metaheuristic methods for Type II instances

For type II instances, the value Δ of MILP1 and MILP2 increas with n up to 6.2% for MILP1 and up to 1.1 % for MILP2. For B&B, except when $n = 10$, all other values Δ are about 1.5%. For TS, except when $n \leq 20$, all Δ is on average 0.4%.

The results MILP2 is the best with $n \leq 40$, MILP1 is the best with $n = 50$. The metaheuristic methods SA is the best when $n \geq 60$ and the result of SA is always much better than TS.

4.4.3 Relation between $\sum N_j$ and $\sum jP_j$

We have seen that there are the relations between two objective functions $\sum N_j$ and $\sum jP_j$ in Section 2.2.1. We now study the relations from a computational experiments point of view.

Because MILP1 is very efficient for solving all the instances of type I for the minimization of $\sum jP_j$, it is possible to determine for each optimal solution, the level of the solution.

We denote by $Level(jP_j)$ the level of the best sequence obtained for the minimization of $\sum jP_j$ and by $Level(N_j)$ the level returned by the best method for the minimization of $\sum N_j$. In Table 4.21, $\Delta(Level(jP_j)/N_j) = \frac{Level(jP_j)-Level(N_j)}{Level(jP_j)}$ is the relative deviation between these two quantities. The column $\#B$ indicates the number of times $Level(jP_j)$ is better than or equal to $Level(N_j)$.

We can see that the best sequence for the problem $\sum jP_j$ has a very good level, interesting for the minimization of $\sum N_j$. In other terms, the use of MILP1 with the objective function $\sum jP_j$ gives a very good solution for the minimization of $\sum N_j$, especially for instances with more than 80 jobs. We notice also that the relative deviation is smaller than 1% for $n = 60$ and 70 jobs, and never greater than 1.7%.

n	$\Delta(Level(jP_j)/N_j)$	# B
10	0.8%	26
20	1.7%	20
30	1.3%	20
40	1.1%	15
50	1.0%	16
60	0.9%	9
70	0.9%	9
80	0.4%	14
90	-1.7%	23
100	-3.5%	29

Table 4.21: Results of the $\Delta(Level(jP_j)/N_j)$ for Type I instances

For type II instance, the results do not show the interest of minimizing $\sum jP_j$ in order to derive good solutions for the minimization of $\sum N_j$.

4.5 Conclusion

This chapter introduces new objective functions based on job positions. The first results that are obtained from these objective functions are summarized in Table 4.22.

For the open problem $1|\widetilde{d}_j|\sum jP_j$ and the strongly $NP-hard$ problem $1|\widetilde{d}_j|\sum w_jP_j$, we propose characteristics, properties and resolution methods.

For the resolution methods, we consider first the exact methods including two kinds of MILP (MILP1 based on Positional variables, and MILP2 based on Linear Ordering variables), a Dynamic Programming algorithm and a Branch-and-Bound algorithm. Heuristic methods BW, FW1, FW2 are also considered for each problem. Two metaheuristic methods, a Tabu Search and a Simulated Annealing have been proposed. The computational experiments of $\sum w_jP_j$ and $\sum jP_j$ have been performed to see the performances of some methods.

Problem	Complexity	
$1\|\beta\|P_{\max}$	$O(1)$	Any solution is optimal, $P_{max} = n$, $\forall\beta$
$1\|\widetilde{d}_j\|P_{\max}$	$O(1)$	$\sigma^{\top}$ is optimal
$1\|\beta\|\sum P_j$	$O(1)$	Any solution is optimal, $\sum P_j = \frac{n(n+1)}{2}$, $\forall\beta$
$1\|\widetilde{d}_j\|\sum P_j$	$O(1)$	$\sigma^{\top}$ is always optimal, $\sum P_j = \frac{n(n+1)}{2}$
$1\|\widetilde{d}_j\|\sum P_j$	$O(1)$	$\sigma^{\top}$ is optimal
$1\|\|\sum jP_j$	$O(1)$	Sequence $\sigma^{\perp}$ is always optimal, $\sum jP_j = \frac{n(n+1)(n+2)}{6}$
$1\|p_j = 1, \widetilde{d}_j\|\sum jP_j$	$O(n\log n)$	BW_{index}
$1\|\widetilde{d}_j, B = 1\|\sum jP_j$	$O(1)$	Simple test
$1\|\widetilde{d}_j, B = 2\|\sum jP_j$	$O(n^3)$	Dynamic programming
$1\|\widetilde{d}_j, B = 2, LPT\|\sum jP_j$	$O(n)$	GR
$1\|\widetilde{d}_j\|\sum jP_j$	open	
$1\|\|\sum w_jP_j$	$O(n\log n)$	sort the jobs in w_j non increasing order
$1\|p_j = 1, \widetilde{d}_j\|\sum w_jP_j$	$O(n\log n)$	BW_{weight}
$1\|\widetilde{d}_j, agree\|\sum w_jP_j$	$O(1)$	Sequence $\sigma^{\top}$ is optimal
$1\|\widetilde{d}_j\|\sum w_jP_j$	Strongly NP-hard	

Table 4.22: The problems related with Positions

Chapter 5

Conclusions and future research direction

The objective of this thesis is to contribute to the characterization of all the optimal solution of the single machine scheduling problem with deadlines. It is well known that this problem may have an exponential number of optimal solutions, and finding their characteristics, instead of having the whole list, is an interesting challenge.

In Chapter 1 we introduce the context of the thesis. We introduce briefly the notion of complexity, we give the required background on resolution methods of operations research and on scheduling theory. Then, we present a survey in the field of characterization of solutions. The chapter terminates with the introduction of the lattice of permutation. This graph structure presents some interesting properties which allow – under some hypotheses – to characterize some sets of solutions. With the support of the lattice of permutation, we identify a new scheduling problem denoted $1|\tilde{d}_j|\sum N_j$, assuming that the jobs are numbered in EDD order and that at least EDD is a feasible solution. The minimization of $\sum N_j$ is equivalent to minimize the level of the feasible sequence searched in the lattice. The first results obtained in this context are summarized and the rest of the structure of the thesis is presented.

In Chapter 2 we give the relations between $\sum N_j$ and many other studies (Kendall's τ distance, crossing number in permutations, One Sided Crossing Minimization problem and Checkpoint Ordering Problem) and many properties are presented. Mathematical expressions of this objective function are given, depending on the type of variables which is used. Finally, the first results that are obtained for this new objective function are given, for some basic problems and for some polynomial particular cases.

In Chapter 3 we propose some resolution methods for this problem. Assuming that the problem is NP-hard, we propose some exponential exact methods: MILP formulations and a B&B algorithm, greedy heuristic algorithms and two metaheuristic methods: a Tabu Search and a Simulated Annealing. Computational experiments are conducted to evaluate the methods.

The study of function $\sum N_j$ in Chapter 2 has highlighted a relation between this function and functions depending on jobs positions. In Chapter 4 we consider the objective

Objective function	The best	MILP1 I	MILP1 II	B&B I	B&B II	BW I	BW II	FW1 I	FW1 II	FW2 I	FW2 II	TS I	TS II	SA I	SA II
$\sum N_j$	among exact			X	X										
	among H						X			X					
	among MetaH												X	X	X
	Overall			X			X							X	
$\sum w_j P_j$	among exact	X	X	X											
	among H								X	X					
	among MetaH													X	X
	Overall	X													X
$\sum jP_j$	among exact	X	X												
	among H						X			X					
	among MetaH														X
	Overall	X													X

Table 5.1: Summarize of the performances of the methods

function $\sum w_j P_j$ where P_j is the position of job J_j. For this new objective function, we study some particular cases, we prove the strongly NP-hardness of the problem and we propose resolution methods. These methods are compared through computational experiments. Finally, the particular case where the weight is equal to the job index is studied. The complexity status remains open, but resolution methods are still proposed and evaluated.

To summarize the results of the methods, we propose the Table 5.1, where a cross indicates the best method. Methods MILP2 and MILP3 are not present because they never really outperform the other methods.

The main ideas for future research directions are the following:

- to generalize the results to other problems in the same category, i.e. which can be solved in polynomial time by the application of a simple rule,
- to continue the characterization, searching for other sequences with a minimum level in the lattice of permutations,
- to close the complexity of the open problems $1|\widetilde{d}_j|\sum N_j$ which is supposed to be NP-hard, and $1|\widetilde{d}_j|\sum jP_j$, which may be solvable in polynomial time.
- to find approximation algorithms for the difficult problem $1|\widetilde{d}_j|\sum w_j P_j$.
- of course, to improve the resolution methods for the difficult problem.

Besides, based on the relations between $\sum N_j$ and many other studies such as Kendall's τ distance, crossing number in permutations, one sided crossing minimization problem and Checkpoint Ordering Problem, there are many interesting results for both of theoretical and practical interest. We can make the link with our problems and find new ideas for the future research directions.

Chapter 6

Appendix 1 - Proof of propositions 6 and 9

We consider here the case where the number of batches is fixed and equal to $B = 2$.

It is assumed here that the jobs in $\mathcal{J} \setminus \mathcal{D}$ (which have the same deadline) are numbered in decreasing processing time order (LPT order). We denote by d the number of jobs in $\mathcal{D}$.

We consider the following algorithm called GR ("Greedy"). We put the jobs of $\mathcal{J} \setminus \mathcal{D}$ in their reverse numbering order: $S = (J_n, J_{n-1}, ..., J_{d+1})$. Then, we insert the jobs of $\mathcal{D}$ consecutively in their reverse numbering order at the last possible position so that $S = (J_n, J_{n-1}, ..., J_k, (J_d, J_{d-1}, ..., J_1), J_{k-1}, ...J_{d+1})$ is the final solution. The algorithm 13 GR describes the method, which runs in $O(n)$.

Algorithm 21 GR (same as Alg. 13)

1: $t = 0$, $k = n$, $S \leftarrow \emptyset$
2: **while** $(t + p_k + \sum_{J_j \in \mathcal{D}} p_j \leq \widetilde{d}_1)$ and $(k \geq d + 1)$ **do**
3: $\quad S \leftarrow S + J_k$
4: $\quad t = t + p_k$, $k \leftarrow k - 1$
5: $S \leftarrow S + (J_d, J_{d-1}, ..., J_1) + (J_{k-1}, J_{k-2}, ..., J_{d+1})$
6: **return** S

Let consider sequence S^{GR} returned by Alg. 13. We denote by $\mathcal{R}$ and $\mathcal{L}$ the sets of jobs which are at the right and at the left of $\mathcal{D}$ in S^{GR}, respectively. We assume that S^{GR} is not optimal and we denote by S^* an optimal sequence. We denote by $\mathcal{R}^*$ and $\mathcal{L}^*$ the sets of jobs which are at the right and at the left of $\mathcal{D}$ in S^*, respectively (see Fig. 6.1 for an illustration).

In S^*, suppose there are x jobs $\mathcal{L}_1 \subseteq \mathcal{L}$ with $\mathcal{L}_1 = \{l_1, l_2, ..., l_x\}$ which are scheduled in $\mathcal{R}^*$, and y jobs $\mathcal{R}_1 = \{r_1, r_2, ..., r_y\} \subseteq \mathcal{R}$ which are scheduled in $\mathcal{L}^*$. We have $p_{l_x} \leq p_{l_{x-1}} \leq ... \leq p_{l_1}$ and $p_{r_y} \leq p_{r_{y-1}} \leq ... \leq p_{r_1}$, and furthermore $p_{l_1} \leq p_{r_y}$. Notice that these jobs are not necessarily consecutive in $\mathcal{L}$ and $\mathcal{R}$.

Because the jobs in $\mathcal{J} \setminus \mathcal{D}$ (which have the same deadline) are numbered in decreasing

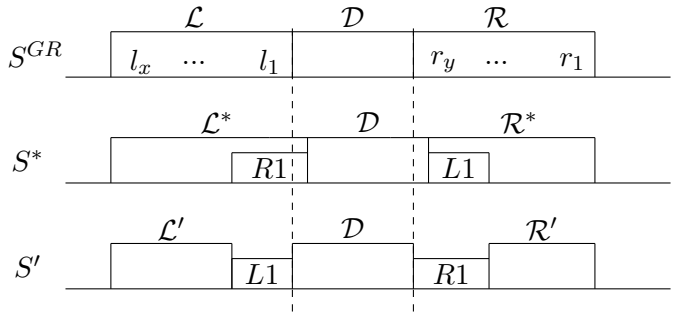

Figure 6.1: Illustration of the notations

processing time order, and $\mathcal{D}$ is ordered at the last possible position, then for any feasible solution (respect of deadlines), we must have:

$$\sum_{j\in\mathcal{R}_1} p_j \leq \sum_{j\in\mathcal{L}_1} p_j$$

Because $p_{l_x} \leq p_{l_{x-1}} \leq ... \leq p_{l_1}$, we have $\sum_{j\in\mathcal{L}_1} p_j \leq x p_{l_1}$. Similarly, because $p_{r_y} \leq p_{r_{y-1}} \leq ... \leq p_{r_1}$, we have $\sum_{j\in\mathcal{R}_1} p_j \geq y p_{r_y}$. Therefore:

$$\begin{aligned}
&\sum_{j\in\mathcal{R}_1} p_j \leq \sum_{j\in\mathcal{L}_1} p_j \\
\Rightarrow &y p_{r_y} \leq x p_{l_1} \\
\Rightarrow &y \leq x \frac{p_{l_1}}{p_{r_y}} \\
\Rightarrow &y \leq x
\end{aligned}$$

because $p_{l_1} \leq p_{r_y}$.

Moreover, since S^* is optimal and S^{GR} is supposed to be not optimal, x must be different to zero (since otherwise, x and y are equal to 0, and S^{GR} and S^* are the same).

Call $L1$ the sequence of x jobs $(l_x, l_{x-1}, ..., l_1)$. Similarly, call $R1$ the sequence of y jobs $(r_y, r_{y-1}, ..., r_1)$. Since S^* has exactly two batches, $L1$ must be in the first positions in $\mathcal{R}^*$, and $R1$ must be in the last positions in $\mathcal{L}^*$.

6.1 Single machine problem, minimization of $\sum N_j$ - Fixed number of batches: $B = 2$

Proposition 10 *(same as Prop. 6) Algorithm 13 GR is optimal for problem $1|\widetilde{d}_j, B = 2, LPT|\sum N_j$. Moreover, $\sum N_j = d.r$, where d and r are the number of jobs in $\mathcal{D}$ and after $\mathcal{D}$, respectively.*

Proof.

Let consider a feasible sequence S' constructed by interchanging in S^* the partial sequences $L1$ and $R1$. Fig.6.1 illustrates this sequence. Notice that this sequence may have more than two batches and that this sequence is feasible (same jobs before $\mathcal{D}$ as in S^{GR}, which is feasible).

We can examine in details the impact of this move on the objective function between S^* and S'. We denote by d the number of jobs in $\mathcal{D}$.

Since the indices of all x jobs in $L1$ are bigger than the indices of the y jobs in $R1$ and of the d jobs in $\mathcal{D}$. We have (see Fig. 6.2):

$$\begin{aligned}\sum N_j(S') &= \sum N_j(S^*) - xy - dx + dy \\ &= \sum N_j(S^*) - xy + d(y - x) \\ &< \sum N_j(S^*)\end{aligned}$$

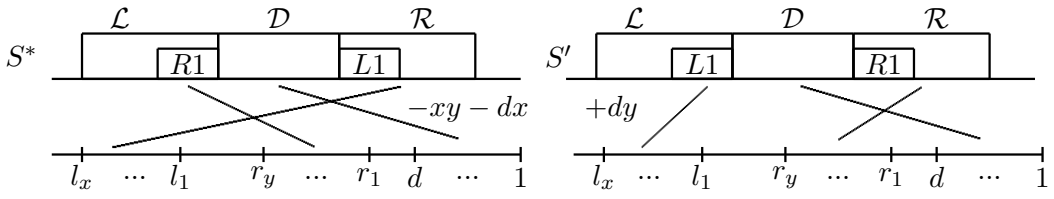

Figure 6.2: Difference between S^* and S'

Because of the definition of S^{GR} and of S', we have: $\sum N_j(S^{GR}) \leq \sum N_j(S')$.

Therefore, we deduce that S^{GR} is optimal.

Moreover, it is clear that we have $\sum N_j = dr$, where d and r are the number of jobs in $\mathcal{D}$ and after $\mathcal{D}$, respectively.

6.2 Single machine problem, minimization of $\sum jP_j$ - Fixed number of batches: $B = 2$

Proposition 11 *(same as Prop. 9) Algorithm 13 GR is optimal for problem* $1|\widetilde{d}_j, B = 2, LPT|\sum jP_j$.

Proof.

In the following, we often using the equality:

$$\sum_{i=a}^{a+b} = (a + b/2)(b + 1) \tag{6.1}$$

We can examine the difference between the objective function for S^* and S'. We have the impact $\Delta = \sum jP_j(S^*) - \sum jP_j(S')$ equal to:

$$\Delta = (d + y)\sum_{i=1}^{x} l_i - (d + x)\sum_{i=1}^{y} r_i$$

We know that $\sum_{i=1}^{x} l_i$ is greater than or equal to the sum of indices of the x last jobs of $\mathcal{L}$. The indices of these jobs are from $d+r+1$ to $d+r+x$. So we have:

$$\sum_{i=1}^{x} l_i \geq \sum_{i=1}^{x} (d+r+i) \tag{6.2}$$

$$\geq \sum_{i=d+r+1}^{d+r+x} i \tag{6.3}$$

$$\geq \Big(d+r+1+\frac{x-1}{2}\Big)x \tag{6.4}$$

applying equality 6.1.

Similarly, $\sum_{i=1}^{y} r_i$ is smaller than or equal to the sum of indices of the y first jobs of $\mathcal{R}$. The indices of these jobs are from $d+1$ to $d+y$. So we have:

$$\sum_{i=1}^{y} r_i \leq \sum_{i=1}^{y} (d+i) \tag{6.5}$$

$$\leq \Big(d+\frac{y+1}{2}\Big)y \tag{6.6}$$

So we have:

$$\begin{aligned}
\Delta &\geq (d+y)\Big(d+r+1+\frac{x-1}{2}\Big)x - (d+x)\Big(d+\frac{y+1}{2}\Big)y \\
&\geq (d+y)\Big(dx+rx+x+x\frac{x-1}{2}\Big) - (d+x)\Big(dy+y\frac{y+1}{2}\Big) \\
&\geq d^2x + drx + dx + dx\frac{x-1}{2} + ydx + yrx + yx + yx\frac{x-1}{2} - d^2y - dy\frac{y+1}{2} - xdy - xy\frac{y+1}{2} \\
&\geq d^2(x-y) + \frac{1}{2}d(x^2-y^2) + \frac{1}{2}d(x-y) + \frac{1}{2}yx(x-y) + rx(d+y)
\end{aligned}$$

We know that $x - y \geq 0$, and $rx(d+y) > 0$, therefore $\Delta > 0$.

We have $\sum jP_j(S') < \sum jP_j(S^*)$. For the same reasons as before (using Hardy's inequality), we deduce that S^{GR} is optimal as well.

Bibliography

[Agnetis *et al.*, 2014] Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli, S. et Soukhal, A. (2014). *Multiagent Scheduling Problems*. Springer. 1.1.3.1, 1.1.3.1, 1.1.4.1

[Allen, 1983] Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM, ACM Press*, pages 832–843. 1.2.1.3

[Aloulou, 2002] Aloulou, M. A. (2002). *Structure flexible d'ordonnancements performances contrles pour le pilotage d'atelier en prsence de perturbations*. Thèse de doctorat, INPL, Nancy. 1.2.1.2, 1.2.1.2

[Aloulou *et al.*, 2004] Aloulou, M. A., kovalyov, M. Y. et Portmann, M.-C. (2004). Maximization problems in single machine scheduling. *Annals of Operations Research*, 129(1-4):21–32. 1.2.1.2

[Aloulou et Portmann, 2003] Aloulou, M. A. et Portmann, M.-C. (2003). An efficient proactive-reactive scheduling approach to hedge against shop floor disturbances. *In* Kendall, G., Burke, E. K., Petrovic, S. et Gendreau, M., éditeurs : *Multidisciplinary Scheduling: Theory and Applications*, pages 223–246, Boston, MA. Springer US. 1.2.1.2

[Argyriou *et al.*, 2010] Argyriou, E., Bekos, M. A., Kaufmann, M. et Symvonis, A. (2010). On Metro-Line Crossing Minimization 1. *Journal of Graph Algorithms and Applications*, 14(1):75–96. 2.1.2

[Artigues, 1997] Artigues, C. (1997). *Ordonnancement en temps reel d'ateliers avec temps de preparation des ressources*. Thèse de doctorat, Paul Sabatier University, Toulouse. 1.2.1.1

[Balas, 1983] Balas, E. (1983). Branch and bound method for traveling salesman problem. *Carnegie Mellon University*. 1.1.3.1

[Bellman, 1957] Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA. 1.1.3.1

[Bennett, 2012] Bennett, K. P. (2012). Crossing minimization within graph embeddings. *Artical*, pages 1–22. 2.1.2

[Benoit et Billaut, 2000] Benoit, M. et Billaut, J.-C. (2000). Characterization of the optimal solutions of the f2//cmax scheduling problem. *In Second Conference on Management Control of Production and Logistics (MCPL 2000)*, Grenoble. 1.2.2.3

[Biedl *et al.*, 2006] Biedl, T., Brandenburg, F. J. et Deng, X. (2006). Crossings and permutations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3843 LNCS:1–12. 2.1.2

[Biedl *et al.*, 2009] Biedl, T., Brandenburg, F. J. et Deng, X. (2009). On the complexity of crossings in permutations. *Discrete Mathematics*, 309(7):1813–1823. 2.1.2, 2.1.2

[Billaut, 1993] Billaut, J.-C. (1993). *Prise en compte des ressources multiples et des temps de prparation dans les problmes d'ordonnancement en temps rel.* Thèse de doctorat, Paul Sabatier University, Toulouse. 1.2.1.1

[Billaut *et al.*, 2012] Billaut, J.-C., Hebrard, E. et Lopez, P. (2012). Complete characterization of near-optimal sequences for the two-machine flow shop scheduling problem. *In* Springer-Verlag, LNCS 7298, L. N. i. C. S. L., éditeur : *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 66–80, Nantes. N. Beldiceanu, N. Jussien, E. Pinson. 1.2.2.1, 1.2.2.3

[Billaut et Lopez, 2011] Billaut, J.-C. et Lopez, P. (2011). Characterization of all r-approximated sequences for some scheduling problems. *In IEEE 16th International Conference on Emerging Technologies and Factory Automation, ETFA, art. no. 6059026*, Toulouse. 1.2.2.3, 2.2.1

[Billaut *et al.*, 2011] Billaut, J.-c., Lopez, P. et Takouti, L. (2011). A method for characterizing the set of optimal sequences for some scheduling problems. *In 12th Congress of the ROADEF, Saint-Etienne, in March 2011.* 1.2.2.3, 2.2.1

[Bixby *et al.*, 2000] Bixby, R., Fenelon, M., Gu, Z., Rothberg, E. et Wunderling, R. (2000). *System Modelling And Optimization*, volume Volume 174. Springer. 1.1.3.1

[Blazewicz *et al.*, 2007] Blazewicz, J., K.H., E., E., P., G., S. et J., W. (2007). *Scheduling computer and manufacturing processes.* Springer, 2rd édition. 1.1.3.1, 1.1.4.1

[Boussaid *et al.*, 2013] Boussaid, I., Lepagnot, J. et Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, 237:82–117. 1.1.3.2

[Bowman, 1959] Bowman, E. H. (1959). The schedule-sequencing problem. *Operations Research*, 7(5):621–624. 1.1.3.1

[Bowman, 1972] Bowman, V. J. (1972). Permutation polyhedra. *SIAM Journal on Applied Mathematics*, 22(4):580–589. 1.2.2.1

[Briand *et al.*, 2005] Briand, C., La, H. T. et Erschler, J. (2005). A new sufficient condition of optimality for the two-machine flowshop problem. *European Journal of Operational Research*, 169:712–722. 1.2.1.3

[Briand *et al.*, 2007] Briand, C., La, H. T. et Erschler, J. (2007). A robust approach for the single machine scheduling problem. *Journal of Scheduling*, 10(3):209–221. 1.2.1.3

[Brightwell et Winkler, 1991] Brightwell, G. et Winkler, P. (1991). Counting linear extensions. *Order*, 8(3):225–242. 1.2.1.2

[Brucker, 2007] Brucker, P. (2007). *Scheduling Algorithms*, volume 93. Springer, 5th édition. 1.1.3.2, 1.1.4.1, 1.1.4.4

[Carlier et Chrétienne, 1988] Carlier, J. et Chrétienne, P. (1988). *Problèmes d'ordonnancement: modélisation, complexité, algorithmes*. Masson. 1.1.4.1

[Chan, 2010] Chan, T. M. (2010). Counting Inversions , Offline Orthogonal Range Counting , and Related Problems. *Canada*, pages 161–173. 2.1.1

[Chen et Bulfin, 1990] Chen, C. L. et Bulfin, R. L. (1990). Scheduling unit processing time jobs on a single machine with multiple criteria. *Computers & Operations Research*, 17:1–7. 4.1.2, 4.2.1, 4.3.2.1

[Chibante, 2010] Chibante, R. (2010). *Simulated Annealing Theory with Applications*. India. 1.1.3.2

[Cook, 1971] Cook, S. A. (1971). The Complexity of Theorem-proving Procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. 1, 1.1.2.1

[Dréo *et al.*, 2006] Dréo, J., Siarry, P., Pétrowski, A. et Taillard, E. (2006). *Metaheuristics for hard optimization: Simulated annealing, tabu search, evolutionary and genetic algorithms, ant colonies*. Springer. 1.1.3.2, 1.1.3.2

[Erschler *et al.*, 1983] Erschler, J., Fontan, G., Merce, C. et Roubellat, F. (1983). A new dominance concept in scheduling n jobs on a single machine with ready times and due dates. *Operations Research*, 31(1):114–127. 1.2.1.3

[Esswein, 2005] Esswein, C. (2005). *Un apport de flexibilit squentielle pour lordonnancement robuste*. Thèse de doctorat, University of Tours, Tours. 1.2.1.1

[Esswein *et al.*, 2005] Esswein, C., Billaut, J. C. et Strusevich, V. A. (2005). Two-machine shop scheduling: Compromise between flexibility and makespan value. *European Journal of Operational Research*, 167(3):796–806. 1.2.1.1

[Garey et Johnson, 1983] Garey, M. R. et Johnson, D. S. (1983). Crossing number is np-complete. *SIAM J. on algebraix and Discrete Methods*, 4(3):312–316. 2.1.2

[Garey et Johnson, 1990] Garey, M. R. et Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA. 1.1.2, 1.1.2.1

[Gendreau et Potvin, 2010] Gendreau, M. et Potvin, J.-Y., éditeurs (2010). *International Series in Operations Research & Management Science Series*. Springer, New York Dordrecht Heidelberg London, 2nd édition. 1.1.3.2

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549. 1.1.3.2

[Glover, 1989] Glover, F. (1989). Tabu Search - Part I. *ORSA journal on Computing*, 2 1(3):4–32. 1.1.3.2

[Glover et Kochenberger, 2003] Glover, F. et Kochenberger, G. (2003). *Handbook of metaheuristics*. Springer. 1.1.3.2

[Glover et Laguna, 1997] Glover, F. et Laguna, M. (1997). *Tabu search*. Springer. 1.1.3.2

[Gonçalves *et al.*, 2016] Gonçalves, T. C., Valente, J. M. S. et Schaller, J. E. (2016). Metaheuristics for the single machine weighted quadratic tardiness scheduling problem. *Computers and Operations Research*, 70:115–126. 1.1.3.2

[Graham *et al.*, 1979] Graham, R. L., Lawler, E. L., Lenstra, J. K. et Kan, A. H. G. R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling. 1.1.4.1

[Hardy *et al.*, 1934] Hardy, G. H., Littlewood, J. E. et Polya, G. (1934). *Inequalities*. Cambridge. 4.2.2.1

[Hartmanis et Leeuwen, 2001] Hartmanis, J. et Leeuwen, J. V. (2001). *Drawing Graphs*. Springer. 2.1.2

[Hungerlaner, 2017] Hungerlaner, P. (2017). The checkpoint ordering problem. *Optimization*, 66(10):1699–1712. 2.1.4

[Jackson, 1955] Jackson, J. (1955). Scheduling a production line to minimize maximum tardiness. *Management Science Research Project*, 43. 1, 2, 1.2.2.2

[Johnson, 1954] Johnson, S. (1954). Optimal Two- and Three-Stage Production Schedules with Setup Times Included. 1, 1.2.2.2

[Jozefowska, 2007] Jozefowska, J. (2007). *Just-in-time scheduling. Models and algorithms for computer and manufacturing systems*. Springer. 1.1.4.1

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. 1.1.2

[Kendall, 1938] Kendall, M. (1938). A new measure of rank correlation. *Biometrika*, 30:81–89. 2.1.1

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gellat, C. D. et Vecchi, M. P. (1983). Optimization by simulated Annealing. *Science*, 220(4598):671–680. 1.1.3.2

[La, 2005] La, H. T. (2005). *Utilisation d'ordres partiels pour la caractrisation de solutions robustes en ordonnancement* . Thèse de doctorat, Université Paul Sabatier, LAAS-CNRS, Toulouse. 1.2.1.3, 1.2.1.3

[Land et Doig, 1960] Land, A. H. et Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520. 1.1.3.1

[Lawler, 1973] Lawler, E. . L. . (1973). Optimal Sequencing of a Single Machine Subject to Precedence Constraints. *Management Science*, 19(5):544–546. 1.1.5.3, 2.3.1, 2.3.2.2

[Le Gall, 1989] Le Gall, A. (1989). *Un systme interactif d'aide la dcision pour l'ordonnancement et le pilotage en temps rel d'atelier.* Thèse de doctorat, Paul Sabatier University, Toulouse. 1.2.1.1

[Lenstra *et al.*, 1977] Lenstra, J. K., Rinnooy Kan, A. H. et Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362. 1.1.5.4, 4.2.1

[Little *et al.*, 1963] Little, J. D. C., Karel, C., Murty, K. G. et Sweeney, D. W. (1963). An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972 – 989. 1.1.3.1

[Maniezzo *et al.*, 2009] Maniezzo, V., Stutzle, T. et Vob, S. (2009). *Metaheuristics: HybridizingMetaheuristics and Mathematical Programming.* Springer. 1.1.3.2

[Manne, 1960] Manne, A. (1960). On the job-shop scheduling problem. *Operations Research*, 8(2):219–223. 1.1.3.1

[Marti et Lanuna, 2003] Marti, R. et Lanuna, M. (2003). Heuristics and meta-heuristics for 2-layer straight line crossing minimization. *Discrete Applied Mathematics*, 127(3): 665–678. 2.1.2

[Mehlhorn et Sanders, 2008] Mehlhorn, K. et Sanders, P. (2008). Generic Approaches to Optimization. *Algorithms and Data Structures: The Basic Toolbox*, pages 233–262. 1.1.3.1

[Metropolis *et al.*, 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. et Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092. 1.1.3.2

[Munoz *et al.*, 2002] Munoz, X., Unger, W. et Vrto, I. (2002). One sided crossing minimization is np-hard for sparse graphs. *LNCS 2265*, pages 115–123. 2.1.3

[Ólafsson, 2006] Ólafsson, S. (2006). Chapter 21 Metaheuristics. *Handbooks in Operations Research and Management Science*, 13(C):633–654. 1.1.3.2

[Osenstiehl, 1963] Osenstiehl, P. R. (1963). Analyse algébrique d'un scrutin. *Mathematics and human sciences*, 4:9–33. 1.2.2.1

[Papadimitriou, 1994] Papadimitriou, D. B. (1994). Computational Complexity. 1.1.2.1

[Pinedo, 2016] Pinedo, M. L. (2016). *Scheduling.* Springer, 5th édition. 1.1.4.1, 1.1.4.4

[Pinot, 2008] Pinot, G. (2008). *Coopération homme-machine pour lordonnancement sous incertitudes.* Thèse de doctorat, University of Nantes, Nantes. 1.2.1.1

[Policella, 2005] Policella, N. (2005). *Scheduling with Uncertainty A Proactive Approach using Partial Order Schedules.* Thèse de doctorat, Universita Degli Studi di Roma La Sapeinza. 1.2.1.2

[Policella *et al.*, 2004] Policella, N., Oddi, A., Smith, S. et Cesta, A. (2004). Generating robust partial order schedules. *In* Wallace, M., éditeur : *Principles and Practice of Constraint Programming-CP 2004: Lecture Notes in Computer Science (LNCS)*, pages 496–511, Toronto, Canada. Springer, Berlin, Heidelberg. 1.2.1.2

[Policella *et al.*, 2005] Policella, N., Oddi, A., Smith, S. F. et Cesta, A. (2005). Generating Robust Partial Order Schedules. *In Principles and Practice of Constraint Programming – CP 2004*, pages 496–511, Berlin, Heidelberg. Springer Berlin Heidelberg. 1.2.1.2

[Rego et Alidaee, 2005] Rego, C. et Alidaee, B. (2005). *Metaheuristic optimization via memory and evolution: tabu search and scatter search*. Kluwer Academic. 1.1.3.2

[Schoute, 1911] Schoute, P. H. (1911). Analytical treatrnent of the polytopes regularly derived frorn the regular polytopes . *Amsterdam*. 1.2.2.1

[Smith, 1956] Smith, W. E. (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66. 1, 1, 1.1.5.4

[Ta et Billaut, 2018a] Ta, T.-T.-T. et Billaut, J.-c. (2018a). Finding a specific permutation of jobs for a single machine scheduling problem with deadlines. *ROADEF2018*. 3.4

[Ta et Billaut, 2018b] Ta, T.-T.-T. et Billaut, J.-c. (2018b). Finding a specific permutation of jobs for a single machine scheduling problem with deadlines. *PMS2018*. 3.4

[Ta *et al.*, 2017] Ta, T.-T.-T., Ringard, K. et Billaut, J.-c. (2017). New objective functions based on jobs positions for single machine scheduling with deadlines. *IESM2017*. 4.2

[Thomas, 1980] Thomas, V. (1980). *Aide à la décision pour lordonnancement datelier en temps réel*. Thèse de doctorat, Université Paul Sabatier, LAAS-CNRS, Toulouse. 1.2.1.1

[T'Kindt et Billaut, 2006] T'Kindt, V. et Billaut, J.-C. (2006). *Multicriteria Scheduling. Theory, Models and Algorithms*. Springer. 1.1.4.1

[Tollis *et al.*, 1999] Tollis, I. G., Battista, G. D., Eades, P. et Tamassia, R. (1999). Graph Drawing. 2.1.2

[Voss *et al.*, 1998] Voss, S., Martello, S., Osman, I. H. et Roucairol, C. (1998). *Advances and Trends in Local Search Paradigms for Optimization*. Springer. 1.1.3.2

[Wagner, 1959] Wagner, H. M. (1959). An integer linear - programming model for machine scheduling. *Stanford University*, pages 131–140. 1.1.3.1

Résumé :

Nous considérons un problème d'ordonnancement à une machine avec dates de fin impratives et nous cherchons caractériser l'ensemble des solutions optimales, sans les énumérer. Nous supposons que les travaux sont numérotés selon la règle EDD et que cette séquence est réalisable. La méthode consiste à utiliser le treillis des permutations et d'associer à la permutation maximale du treillis la séquence EDD. Afin de caractériser beaucoup de solutions, nous cherchons une séquence réalisable aussi loin que possible de cette séquence. La distance utilisée est le niveau de la séquence dans le treillis, qui doit être minimum (le plus bas possible). Cette nouvelle fonction objectif est étudiée. Quelques cas particuliers polynomiaux sont identifiés, mais la complexité du problème général reste ouverte. Quelques méthodes de résolution, polynomiales et exponentielles, sont proposées et évaluées. Le niveau de la séquence étant en rapport avec la position des travaux dans la séquence, de nouvelles fonctions objectifs en rapport avec les positions des travaux sont identifiées et étudiées. Le problème de la minimisation de la somme pondérée des positions des travaux est prouvé fortement NP-difficile. Quelques cas particuliers sont étudiés et des méthodes de résolution proposées et évaluées.

Mots clés :

ordonnancement, une machine, dates impératives, treillis, caractérisation, complexité

Abstract :

We consider a single machine scheduling problem with deadlines and we want to characterise the set of optimal solutions, without enumerating them. We assume that jobs are numbered in EDD order and that this sequence is feasible. The key idea is to use the lattice of permutations and to associate to the supremum permutation the EDD sequence. In order to characterize a lot of solutions, we search for a feasible sequence, as far as possible to the supremum. The distance is the level of the sequence in the lattice, which has to be minimum. This new objective function is investigated. Some polynomially particular cases are identified, but the complexity of the general case problem remains open. Some resolution methods, polynomial and exponential, are proposed and evaluated. The level of the sequence being related to the positions of jobs in the sequence, new objective functions related to the jobs positions are identified and studied. The problem of minimizing the total weighted positions of jobs is proved to be strongly NP-hard. Some particular cases are investigated, resolution methods are also proposed and evaluated.

Keywords :

scheduling, single machine, deadlines, positions, lattice, characterization, complexity