Theoretical and practical contributions to star observation scheduling problems

Florian Fontan

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques et Informatique**
Arrêté ministériel : 25 mai 2016

Présentée par

Florian FONTAN

Thèse dirigée par Nadia BRAUNER (Professeur, UGA)
et codirigée par Pierre LEMAIRE (MCF, Grenoble INP)

préparée au sein du **Laboratoire G-SCOP**
dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Contributions théoriques et pratiques à l'ordonnancement d'observations d'objets célestes

☆ ☆ ☆

Theoretical and practical contributions to star observation scheduling problems

Thèse soutenue publiquement le **21 novembre 2019**, devant le jury composé de :

Monsieur **Christophe Rapine** *(Rapporteur)*
Professeur, Université de Lorraine

Madame **Safia Kedad-Sidhoum** *(Rapportrice)*
Professeure, Cnam

Monsieur **Christian Artigues** *(Président du jury)*
Directeur de Recherche, LAAS-CNRS

Madame **Anne-Marie Lagrange** *(Examinatrice)*
Directrice de Recherche, IPAG-CNRS

Madame **Nadia Brauner** *(Directrice de thèse)*
Professeure, Université Grenoble Alpes

Monsieur **Pierre Lemaire** *(Encadrant)*
Maître de Conférence, Grenoble INP

This page is intentionally left blank.

Remerciements

Cette thèse n'aurait tout d'abord bien évidemment pas été possible sans mes encadrants Nadia Brauner et Pierre Lemaire. Je les remercie d'avoir cru en moi et de m'avoir soutenu pendant ces trois années, J'ai grâce à eux passé trois superbes années, autant sur le plan scientifique que sur le plan personnel. Je les remercie d'avoir toujours été présents quand j'en avais besoin. Je les remercie également de tout ce qu'ils m'ont apporté au niveau de l'enseignement, de m'avoir laissé participer au challenge EURO/ROADEF et de m'avoir donné l'opportunité de passer quatre mois à Minsk.
Je remercie Safia Kedad-Sidhoum et Christophe Rapine d'avoir accepté d'être rapporteurs de ma thèse, et pour Christophe Rapine, également d'avoir participé au Comité de Suivi Individuel pendant ma thèse. Je remercie Christian Artigues d'avoir présidé mon jury de thèse et Anne-Marie Lagrange d'avoir accepté d'être examinatrice de ma thèse.
Je remercie encore Anne-Marie Lagrange d'avoir proposé cette collaboration entre les équipes de Recherche Opérationnelle et d'Astrophysique de Grenoble, ainsi que Pascal Rubini, l'ingénieur qui a développé le logiciel SPOT utilisé par les astrohpysiciens et avec qui j'ai régulièrement échangé durant ma thèse.
Je remercie bien-sûr Luc Libralesso de s'être inscrit au challenge, Vincent Jost de m'avoir poussé à le rejoindre, et encore Luc d'avoir accepté et de m'avoir appris tant de choses depuis.
Я хочу поблагодарить Михаила Ковалева за то, что он принимал меня четыре месяца в Минске. Из Минска я также хочу поблагодарить Максима Баркетова, Якова Шафранского, Александра Ковалева и всю административную команду Объединенного института проблем информатики Национальной академии наук Беларуси за их помощь, в частности Евгения Ефимова.
Je remercie tout le personnel du laboratoire G-SCOP, le service informatique, le service financier ainsi que le personnel administratif qui permettent aux doctorants d'effectuer leur thèse dans des conditions optimales ; et l'ensemble des membres du laboratoire, de l'association des doctorants et les autres doctorants qui n'ent font pas partie, qui font de ce lieu un endroit convivial.
Pour m'avoir aidé sur les questions techniques en informatique et en mathématiques, je remercie Nicolas, Denis et Amaury.
Je remercie également mes amis et ma famille.

This page is intentionally left blank.

Introduction

To a large extent, this thesis is the result of a collaboration between the Operations Research team at G-SCOP laboratory and the IPAG research unit (Institute for Planetary sciences and Astrophysics, Grenoble) of Grenoble's Universe Sciences Observatory (OSUG).
Astrophysicists want to schedule observations on a telescope and, for each possible target (star), there exist time-windows when it is visible, a required duration for its observation, and an interest for observing it; the objective is to maximize the total interest of the schedule. The primary goal of this thesis was to improve the algorithms implemented in the software used by the astrophysicists, while integrating additional specific constraints and assessing the potential gains of introducing some flexibility in the model.

This flexibility is related to observation duration. Indeed, shortening an observation leads to a picture of lesser quality. However, it would be worth doing if that makes room for another one. More generally an observation remains relevant even if its observation time is slightly less than the required value.
In Chapter I, we study this aspect from a theoretical point of view. We consider scheduling problems where jobs have a variable processing time: one can *decide* the processing time of each job. The profit for a job then depends on its allocated processing time and the objective is to maximize the total profit of the schedule. Our purpose is to draw the line between P-easiness and NP-hardness for various profit functions and machine environments, with an emphasis on models leading to the practical star scheduling problem.

Then, in Chapter II, we take care of the practical problem resolution. We define the complete model and propose several ways of integrating the flexible observation durations in it. We develop a Large Neighbourhood Search algorithm for each variant. It reaches the required practical criteria from simplicity and adaptability to speed and efficiency. Its performances are compared to previous approaches and the gain from flexibility is estimated.

Chapter III is not related to the star observation scheduling problem. Every other year, the French Operational Research and Decision Support Society (ROADEF) jointly with the European Operational Research Society (EURO) organizes an optimization challenge open to everyone. Each edition is a collaboration with an industrial partner, and the problems and instances submitted correspond to real life industrial optimization problems. Previous

challenges have been organized in collaboration with Air Liquide (2016), SNCF (2014), Google (2012), EDF (2010)... The 2018 edition of the challenge was dedicated to a cutting optimization problem in collaboration with Saint-Gobain.
In Chapter III, we present the heuristic *Branch and Bound* algorithm we submitted for the final phase of the challenge. We describe the *Branching scheme* including dominance criteria and symmetry breaking strategy; and the anytime *Tree search* algorithm called MBA* which is inspired from the classical $A*$ and *Beam search* algorithms.

The three chapters can be read independently. Therefore, Because of the unrelatedness of cited papers between them, we choose to split the bibliography into each chapter.
While Chapters I and II have been written with my supervisors Nadia Brauner and Pierre Lemaire, Chapter III has been written together with Luc Libralesso.

Table of Contents

This page is intentionally left blank.

Chapter I

Complexity of processing time dependent profit maximization scheduling problems

In this chapter, we study the complexity of scheduling problems where jobs have a variable processing time: one can *decide* the processing time of each job. The profit for a job then depends on its allocated processing time.

In our experience, the problem originates from astrophysics and the search for exoplanets (Lagrange et al. 2016). Astrophysicists want to schedule observations on a telescope and, for each possible target (star), there exist time-windows when it is visible, a required duration for its observation, and an interest for observing it; the objective is to maximize the total interest of the schedule. This primary version of the problem has been described and solved by Catusse et al. 2016, but it appears that shortening an observation would be worth doing if that makes room for another one. More generally an observation remains relevant even if its processing time is slightly less than the required value, with an accordingly downgraded interest. Such a situation can be modelled by processing time dependent profits: hence there is a need to study such models and, first of all, their complexity (this chapter); the practical resolution of the complete problem is the subject of Chapter II.

Therefore, in this chapter, we first state formally the problem and propose several profit functions (Section I.1) together with a literature review (Section I.2); then, we prove some generic properties (Section I.3) moving on to NP-completeness results (Section I.4) before considering polynomial cases (Section I.5).

I.1 Processing time dependent profit

We consider a scheduling problem with n jobs; each job T_j has a deadline d_j and a profit function $w_j(p_j)$ that depends on the decided processing time p_j.

The objective is to maximize the total profit:

$$\max \sum_{j=1}^{n} w_j(p_j)$$

with the convention that $p_j = 0$ if T_j is not scheduled. Figure I.1 shows three examples of profit functions. The first one depicts how in a classical job selection problem, the profit $\bar{w}_j$ of a job T_j can be modeled by a processing time dependent profit function where $w_j(p) = 0$ if the processing time does not reach the required one and $w_j(p) = \bar{w}_j$ otherwise. The second example is a linear profit and the third one represents the profit of a star observation.

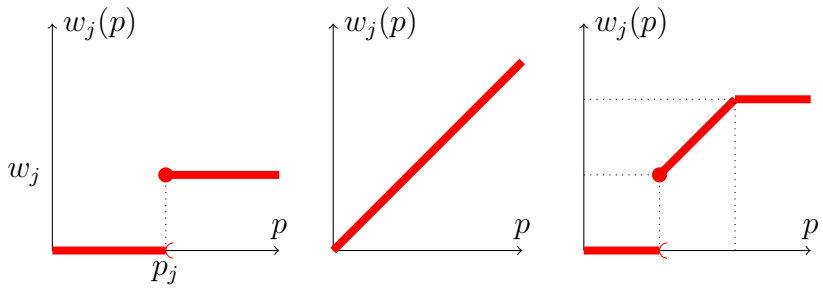


Figure I.1 – Examples of profit functions of: (a) a classical scheduling problem ($w_j(p) = w_j$ if $p \geq p_j$, 0 otherwise); (b) basic (linear profit) problem; (c) the star observation problem

Note that, if there are no release dates (all jobs are available at $t = 0$), a job can always be scheduled with a null processing time at the beginning of the schedule. However, when jobs have different release dates, it may be impossible to schedule all jobs in an optimal solution, even with null processing time. This is illustrated on Figure I.2. In this example, we assume that the profit yielded by J_1 is much greater than the profit that would yield J_2 if it were scheduled and that preemption is not allowed. If one schedules J_2, then J_1 has to be interrupted, and therefore no optimal solution exists where J_2 is scheduled. However, the feasibility can always be ensured since not scheduling any job is a feasible solution.
Also note that the objective function is not a regular criterion and the instance size (as input of a Turing machine) depends on how w_j are defined. In the remainder of this chapter, we call this problem and its variants Processing time Dependent Profit Scheduling Problems (PDPSP).

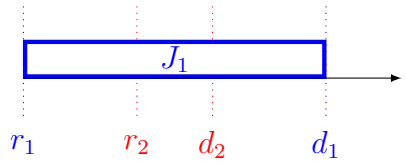


Figure I.2 – Example of an instance where all jobs cannot be scheduled in an optimal solution

We propose now several profit functions that are studied thereafter; to this extent, we adapt the 3-field notation (see e.g. Pinedo 2015) as follows:

$$\alpha \mid f, \beta \mid -\sum w_j(p_j)$$

where, in addition to the classical α and β, parameter f describes the profit function. We note the objective $-\sum w_j(p_j)$ as the minimization of the negative profit, since, in scheduling, objectives are usually to be minimized (Pinedo 2015). Also note that deadlines are not explicitely indicated, since they are implied by the objective.
In this chapter, we consider the profit functions presented in Figure I.3. The one corresponding to the original astrophysics problem is the last. The other ones are simplifications that allow to understand the role of each of its components. The mnemonic for the abbreviations is: ST stands for Setup Time, B for Bounded, L for Linear, and IP for Initial Profit. In all cases, the parameter b_j is called the growth rate of job T_j, $w_j^{\min}$ its minimum profit, $w_j^{\max}$ its maximum profit, $p_j^{\min}$ its minimum processing time and $p_j^{\max}$ its maximum processing time. Note that these denominations are slightly abusive: indeed, the minimum processing time of any job T_j is 0 and the maximum processing time is d_j, but there is no use scheduling T_j during $p_j \in \left]0, p_j^{\min}\right[\cup \left]p_j^{\max}, d_j\right]$. We also define $b_{\max} = \max_j b_j$ and $d = \max_j d_j$.
In Section I.5, we present polynomial algorithms that generalize to a Piecewise Linear Profit (PLP) model. In this model, each job T_j has K_j pieces. A piece is given by an initial processing time p_j^k, an inital profit w_j^k and a growth rate b_j^k:

$$w_j(p) = w_j^k + b_j^k(p - p_j^k) \qquad \text{for } p_j^{k-1} \leq p < p_j^k$$

and we assume that:

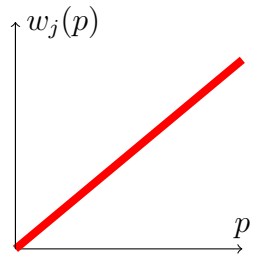
(1) $p_j^0 = 0$ and $w_j^0 = 0$ (no profit for a null processing time);

(2) $p^{K_j} \geq d_j$ (profit must be defined at least up to the deadline);

(3) $w_j^{k+1} \geq w_j^k + b_j^k(p_j^{k+1} - p_j^k)$ (profit is non-decreasing);

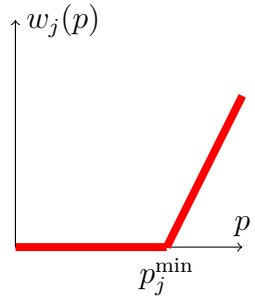
Linear Profit (LP)

$$w_j(p) = b_j p$$



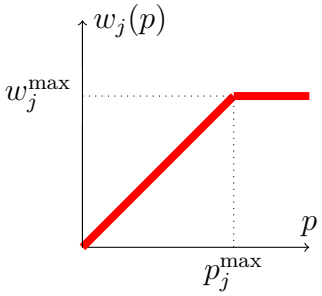
Linear Profit with Setup Time (LPST)

$$w_j(p) = \begin{cases} 0 & \text{for } p < p_j^{\min} \\ b_j(p - p_j^{\min}) & \text{for } p \geq p_j^{\min} \end{cases}$$



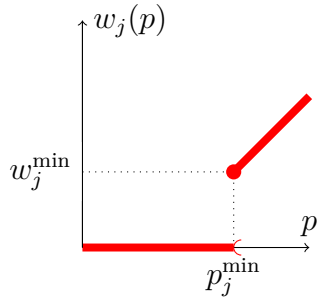
Linear Bounded Profit (LBP)

$$w_j(p) = \begin{cases} b_j p & \text{for } p < p_j^{\max} \\ b_j p_j^{\max} & \text{for } p \geq p_j^{\max} \end{cases}$$



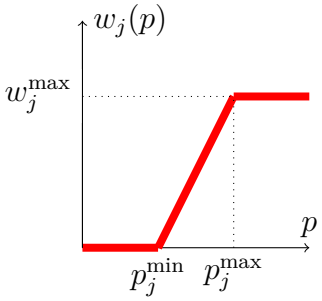
Linear Profit with Setup Time and Initial Profit (LPSTIP)

$$w_j(p) = \begin{cases} 0 & \text{for } p < p_j^{\min} \\ w_j^{\min} + b_j(p - p_j^{\min}) & \text{for } p \geq p_j^{\min} \end{cases}$$



Linear Bounded Profit with Setup Time (LBPST)

$$w_j(p) = \begin{cases} 0 & \text{for } p < p_j^{\min} \\ b_j(p - p_j^{\min}) & \text{for } p_j^{\min} \leq p < p_j^{\max} \\ b_j(p_j^{\max} - p_j^{\min}) & \text{for } p \geq p_j^{\max} \end{cases}$$



Linear Bounded Profit with Setup Time and Initial Profit (LBPSTIP)

$$w_j(p) = \begin{cases} 0 & \text{for } p < p_j^{\min} \\ w_j^{\min} + b_j(p - p_j^{\min}) & \text{for } p_j^{\min} \leq p < p_j^{\max} \\ w_j^{\min} + b_j(p_j^{\max} - p_j^{\min}) & \text{for } p \geq p_j^{\max} \end{cases}$$

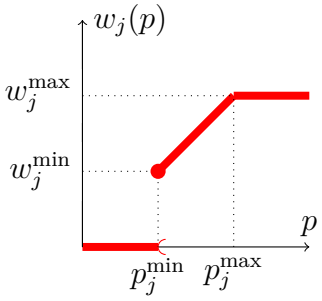


Figure I.3 – Profit functions

(4) p_j^k and w_j^k are integers, b_j^k are rationnals.

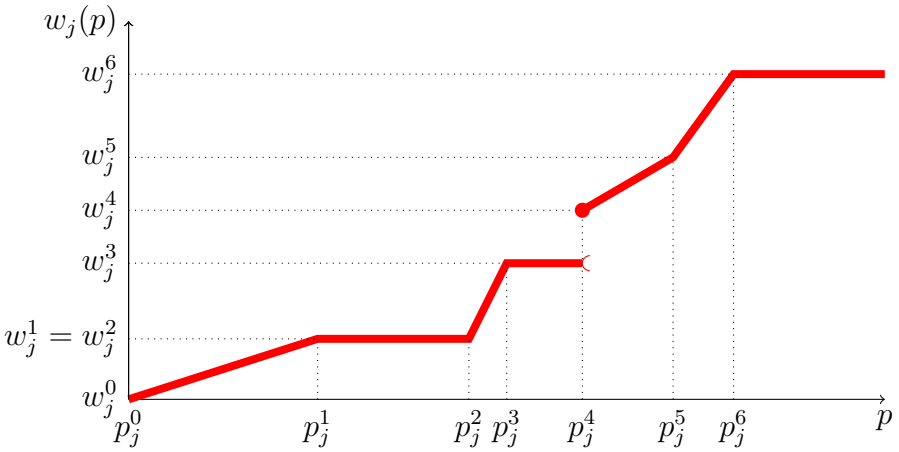


Figure I.4 – A PLP function showing the different types of pieces

A PLP can be any non-decreasing piecewise-linear function starting at 0 and may be neither continuous nor concave; Figure I.4 shows the different possible configurations of pieces. Note that, for a given p, computing $w_j(p)$ requires $O(\log K_j)$ time to search for the piece k such that $p_j \in [p_j^k, p_j^{k+1}[$. The assumption that input data are integers holds for every model (and is further discussed later, see Section I.3). For LBP, LBPST and LBPSTIP, we allow growth rates to be rationnal, but such that $w_j^{\max}$ remain integers.
For a PDPSP with a PLP, we note $K = \max_{j=1,\dots,n} K_j$.

A general PDPSP is obviously an NP-complete problem. In this chapter, we study the complexity of the various PDPSP with a focus on the PLP model in order to find easier cases.

I.2 Literature review and applications

From a theoretical point of view, PDPSP can been seen as a combination of job selection (also called scheduling with rejection) and scheduling with controllable processing times.
Job selection is a classical class of scheduling problems. A review of these problems can be found in Slotnick 2011 and Shabtay et al. 2013. The most basic variants of job selection problems are the Knapsack Problem and the scheduling problem $1 \mid\mid \sum U_j$. Research focuses on richer variants of the problem. For example, Bartal et al. 2000; Zhang et al. 2009 consider a job

selection problem of which objective is to minimize the sum of the makespan and the penalties of all rejected jobs while Engels et al. 2003 minimize the sum of the weighted completion times of the jobs scheduled plus the sum of the penalties of the jobs rejected. Zhang et al. 2010 minimize any regular objective function while bounding the sum of the penalties of the rejected jobs. Shabtay et al. 2012 propose a bicriteria approach, the first objective being a regular criterion and the second one being the total rejection cost. Lin and Ying 2015 study of permutation flowshop with job rejection. Eun et al. 2017 study a completion time dependent profit maximization scheduling problem, while in PDPSP, the profit depends on the processing time.
Scheduling problems with controllable processing times have also received great attention (see Shabtay and Steiner 2007; Shioura et al. 2018 for surveys). In those problems, processing times are controllable by allocating a resource to job operations. The objective is then either related to job completion times, resource consumption cost or a combination of both. When there is no discontinuity in the profit function, algorithms developed for scheduling problems with controllable processing times may also solve PDPSP. For example, the problem $1 \mid \text{LBP} \mid -\sum w_j(p_j)$ is equivalent to the scheduling problem with controllable processing times subject to deadlines adressed in Janiak and Kovalyov 1996.
A variant of PDPSP has been introduced by Cao et al. 2006. They studied it as a bi-criteria problem, minimizing the weighted number of rejected jobs and minimizing the total cost of compression. They prove the NP-hardness of the problem, and propose a FPTAS, a pseudo-polynomial algorithm and a greedy heuristic. In this chapter, we aggregate both criteria, which allows us to study more accurately and exclusively the complexity of those problems.
PDPSP are also related to scheduling problems with late work criteria, noted Y_w (see Sterna 2011 for a survey), in which only the units executed after the due dates are penalized. For example, as used in Section I.9.3, $P \mid \text{LBP}, \text{pmtn}, r_j \mid -\sum w_j(p_j)$ can be seen as the late job problem $P \mid \text{pmtn}, r_j \mid \sum w_j Y_j$, that can be solved with a minimum cost flow model in polynomial time (Leung et al. 2004).

PDPSP appear as subproblems of several applications. As stated in the introduction, the issue arises in astrophysics and the search for exoplanets (Lagrange et al. 2016) in which the picture quality depends on the observation time; even though the current practice does not take advantage of such a possibility, due to the lack of tools to handle it.
A similar phenomenon appears in Earth observation scheduling problems. In the problem described by Verfaillie and Lemaître 2001, polygon acquisitions may be only partially satisfied. The gain associated with a partially sat-

isfied request is either linearly or convexly dependent of the useful aquired surface. Cordeau and Laporte 2005; Habet et al. 2010; Wang et al. 2011; Tangpattanakul et al. 2015 used the same model for variants of the problem. Yang and Geunes 2007 also introduced a scheduling problem including PDPSP as subproblem. As in PDPSP, the objective is profit maximization, but their model also includes tardiness costs. They propose two heursitics with computational experiments on randomly generated instances.

I.3 Notations and general results

Some notations are used consistently throughout the rest of this chapter. I denotes an instance and $|I|$ is its size (as input of a Turing machine). S denotes a solution, $|S|$ is the number of tasks included in S (possibly with a null processing time). For a task $T_j \in S$: $p_j(S)$ denotes its processing time, $s_j(S)$ its start date and $C_j(S)$ its completion time (end date). When there is no ambiguity on the solution, we will just write p_j, s_j and C_j. $w(S)$ denotes the total profit of S (*i.e.*, $w(S) = \sum_{T_j \in S} w_j(p_j)$). $\mathcal{U}$ denotes a dominant set of solutions, *i.e.* for any instance, there exists an optimal schedule in $\mathcal{U}$.
Furthermore, when considering complexity issues and thus decision problems, we shall add a bound $W \in \mathbf{N}$ and ask whether there exists a feasible schedule such that $\sum_j w_j(p_j) \geq W$. For all the cases considered in this chapter, providing start dates, processing times and, if needed, processing machines for all tasks completely defines a solution; for each case, such a solution is indeed a polynomial certificate and will be enough to straightforwardly conclude that it belongs to NP.

The first remarkable fact about processing time dependent profit scheduling problems, is that the profit functions proposed in Section I.1 strongly relate one to another, and there is indeed a whole complexity hierarchy that exists among them and is represented by Figure I.5. This hierarchy is useful to exhibit maximally-polynomial or minimally-NP-complete cases.
Then, a generic dominance property exists:

Lemma I.3.1
For $P\,|\,|\,-\sum w_j(p_j)$, the set of solutions for which all tasks are scheduled (possibly with $p_j = 0$) and such that, on each machine, tasks are scheduled in non-decreasing order of their deadlines and without idle time, is dominant.

Proof. In any solution, idle times can always be removed and tasks that are not scheduled can always be added at time 0 with a null processing time,

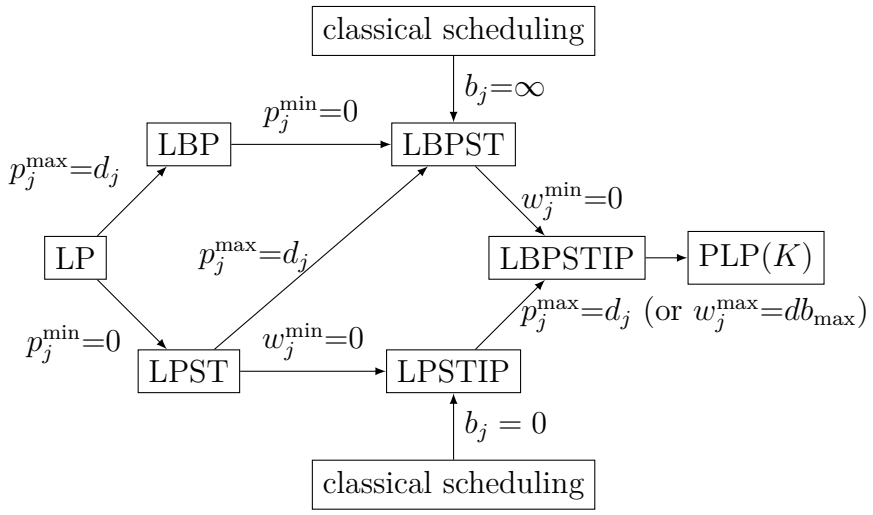
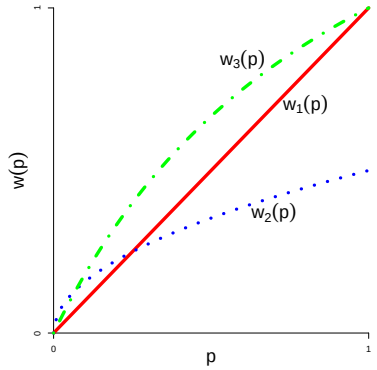


Figure I.5 – Complexity hierarchy of PDPSP problems. An arc from A to B means that A is a special case of B, and a label indicates the particular settings of B that corresponds to A. $PLP(K)$ is the particular case when the number of pieces is bounded by K.

without altering the feasibility nor the value of the solution. Hence, we can consider, without loss of generality, that all tasks are scheduled and without idle time.
Then, the proof relies on a classical exchange argument. Let S be an optimal solution. If there exist unordered pairs of tasks (that is (T_j, T_k) such that T_j is scheduled after T_k on the same machine whereas $d_j < d_k$), then there exists an unordered pair (T_j^*, T_k^*) such that T_j^* is scheduled immediately after T_k^*. The solution obtained by exchanging T_j^* and T_k^* remains feasible and optimal, while strictly decreasing the number of unordered pairs. Repeating this operation leads to a feasible optimal solution without any unordered pairs, *i.e.* with tasks scheduled in non-decreasing order of their deadlines. ■

This lemma is very useful but is not enough to directly provide optimal solutions, as one does not know how long each task must be executed. Still, note that for the single machine case, a solution can be characterized by the list of the processing times allocated to each task.

In scheduling, it is quite common that start dates and end dates are integers as soon as input data are integers and there is no preemption. So one could think that similar results could be designed for all but exotic profit functions. This is not that simple and one should be aware that tasks may end at non-integer (even irrational) instants as soon as the profit function is not linear



Three tasks T_1, T_2, T_3 such that :

- $w_1(p) = p$
- $w_2(p) = \sqrt{p}/2$
- $w_3(p) = 2 - 2/(1+p)$

and $d_1 = d_2 = d_3 = 1$

Example 1: $I = \{T_1, T_2\}$. *In an optimal solution:* $p_1 = 15/16$, $p_2 = 1/16$.
Example 2: $I = \{T_1, T_3\}$. *In an optimal solution:* $p_1 = \sqrt{2} - 1$, $p_3 = 2 - \sqrt{2}$.

Figure I.6 – Two simple examples of non-linear PDPSP for which an optimal solution must have some tasks with non-integer start or end dates.

(simple such cases are depicted in Figure I.6). However, for the PLP model, as we assume integer data and (piecewise) linear profit, solutions with integer processing times are indeed dominant:

Lemma I.3.2
For $P \mid \text{PLP} \mid -\sum w_j(p_j)$, the set of solutions such that:

(1) all tasks are scheduled (possibly with $p_j = 0$) and on each machine, tasks are scheduled in non-decreasing order of their deadlines without idle time;

(2) all start dates and processing times are integers, *i.e.* for all $j = 1, \ldots, n$: $s_j, p_j \in \{0, \ldots, d_j\}$

is dominant.

Proof. (1) has already been proven by Lemma I.3.1.
For (2), let us first remind that $d_j \in \mathbf{N}$ and $p_j^k \in \{0, \ldots, d_j\}$ for all $j = 1, \ldots, n$ and $k = 1, \ldots, K_j$. In any solution, idle times can be removed and the last task on a machine can always be extended up to its deadline (possibly with no profit). As a consequence, one can assume without loss of generality that on each machine: the first task starts at time 0, every other task starts as soon as the preceding one ends, and the last task ends at an integer time (its deadline). Clearly, if all processing times are integers, then all start dates and end dates are integers. So, let's prove that there always exists an optimal solution with only integer processing times.

If that is not true, then there exists a solution S which has the above properties, which is optimal, and which is "minimal", in the sense that it has the minimum number of tasks with non-integer processing times. Let T_{j_1} be the first task such that $p_{j_1} \notin \mathbf{N}$; as the total processing times on a machine sums up to an integer, there exists another task T_{j_2}, on the same machine, such that $p_{j_2} \notin \mathbf{N}$; we assume T_{j_2} to be the first such task after T_{j_1}. There exists k_1 such that $p_{j_1}^{k_1} \leq \lfloor p_{j_1} \rfloor < p_{j_1} < \lceil p_{j_1} \rceil \leq p_{j_1}^{k_1+1}$ and there exists k_2 such that $p_{j_2}^{k_2} \leq \lfloor p_{j_2} \rfloor < p_{j_2} < \lceil p_{j_2} \rceil \leq p_{j_2}^{k_2+1}$.
If $b_{j_1} = b_{j_2}$, then p_{j_1} can be decreased, and p_{j_2} increased by the same amount, until either p_{j_1} or p_{j_2} is an integer (which then contradicts that S is "minimal"). If $b_{j_1}^{k_1} < b_{j_2}^{k_2}$ then there exists $\epsilon > 0$ such that decreasing p_{j_1} by ϵ and increasing p_{j_2} by the same amount yields a feasible and strictly better solution, which contradicts the optimality of S; the reverse holds if $b_{j_1}^{k_1} > b_{j_2}^{k_2}$.
In this case, note that T_{j_1} and T_{j_2} being the first two tasks with non-integer durations, none of the tasks in between can end on an integer date, and in particular, none ends at its deadline; hence it is indeed always possible to increase p_{j_1} by some $\epsilon > 0$ and decrease p_{j_2} by the same amount, without altering the feasibility of the solution. ■

Lemma I.3.2 still does not provide a direct optimal solution, but we can develop a dynamic programming algorithm on it:

Theorem I.3.1
Algorithm 1 returns an optimal solution to $P \mid \text{PLP} \mid -\sum w_j(p_j)$ in time $O(n(md^{m+1} + \log n))$.

Proof. Algorithm 1 is a standard dynamic-programming algorithm. The value $f^*(j, C)$ is the best value that can be obtained by scheduling the first j tasks so that the machines end at the completion times defined by C.
By Lemma I.3.2, there exists an optimal solution with tasks ordered as in step 1. Thus, it is enough to consider the tasks in this order and to enumerate, for a task T_j, all possible end dates on all machines. This is done by step 3: the base case is straightforward, whereas the two next cases forbid idle time (since there always exists an optimal solution without idle time) and the general case performs the actual enumeration.
The complexity of step 1 is $O(n \log n)$. Then, the complexity of step 2 is $O(nd \log K)$ which is $O(nd^2)$ since $K \leq d$; this pre-computing enables to get $w_j(C_i - s)$ in $O(1)$ during step 3. The size of f^* is nd^m and it takes $O(md)$ to compute a given value in step 3. The overall complexity is thus $O(nmd^{m+1} + n \log n)$. ■

Algorithm 1 $P \mid \text{PLP} \mid -\sum w_j(p_j)$

INPUTS: an instance I

1: Sort tasks in non-decreasing order of their deadlines
2: Precompute $w_j(p)$ for all task T_j, $j = 1, \dots, n$ and for all $p = 0, \dots, d_j$
3: Let $\mathcal{C} = \{C \in \{0, \dots, d\}^m, C_1 \leq \dots \leq C_m\}$. Return $\max_{C \in \mathcal{C}} f^*(n, C)$ such that:

$$f^*(j, C) = \begin{cases} 0 & \forall j = 0, \dots, n \text{ and } C = \{0, \dots, 0\} \\ -\infty & \text{if } j = 0 \text{ and } C \neq \{0, \dots, 0\} \\ & \textit{(idle time at the beginning is never necessary)} \\ -\infty & \text{if } d_j < \max C_i \\ & \textit{(no task can end at } C_i\textit{)} \\ \max\limits_{\substack{1 \leq i \leq m \\ s \in \{0, \dots, C_i\}}} & f^*(j-1, C - (C_i - s)e^i) + w_j(C_i - s) \\ & \text{otherwise} \end{cases}$$

where C is a m-dimensional vector containing the required completion time of the last task scheduled on each machine and e^i is the vector such that $e^i_i = 1$ and $e^i_k = 0$ for all $k \neq i$.

Note that Algorithm 1 runs in pseudo-polynomial time if the number of machines is fixed and even in polynomial time if d is also fixed (or more generally if either due-dates or processing times are bounded). That is: Algorithm 1 runs in a reasonable time if the numbers involved are reasonable. This is a standard property for a number-problem (well-known for, e.g., Partition or Subset Sum) which may actually apply in practice. For instance, in the star scheduling problem, the duration of a night is naturally bounded and the duration of an observation is technically limited, providing practical upper bounds on respectively due-dates and processing times.

Lemma I.3.2 provides a first overview of the structure of solutions with a PLP function. Still, we can have better results: most tasks can be scheduled during a p^k_j, and only a few will not. If two consecutive tasks are not scheduled during a p^k_j, one of them can have its processing time decreased for the benefit of the other one (the increase of the first task may be limited by its deadline).

We get even stronger results if we only consider "interesting" processing times instead of all p^k_j. We define $\mathcal{P}_j$ the set of "interesting" processing times of

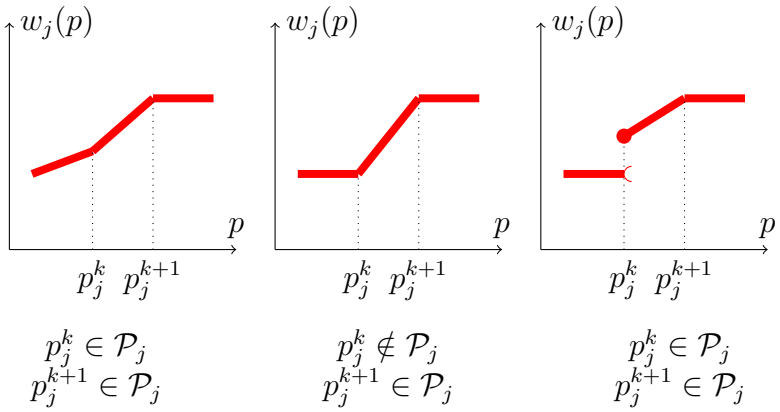


Figure I.7 – Examples of "interesting" and not "interesting" processing times.

task T_j as follows:

$$\mathcal{P}_j = \left\{\rho_j^1, \dots, \rho_j^{K_j^\rho}\right\} = \left\{\rho, \quad \exists k,\ \rho = p_j^k \text{ and } w_j^k(\rho - 1) < w_j^k(\rho)\right\}$$

with $\rho_j^1 \leq \dots \leq \rho_j^{K_j^\rho}$ and we call T_j a *regular* task if $p_j \in \mathcal{P}_j$ and a *singular* task if $p_j \notin \mathcal{P}_j$ (note that $K_j^\rho \leq K_j$).
An "interesting" processing time corresponds to a p_j^k except when the previous piece has a null growth rate and there is no discontinuity between them. The need to define $\mathcal{P}_j$ is illustrated in Figure I.7; for instance, $p_j^{\min}$ is "interesting" for LPSTIP or LBPSTIP, but it is not for LPST and LBPST. The following lemma indicates what makes those processing times "interesting":

Lemma I.3.3
For $P \mid \text{PLP} \mid -\sum w_j(p_j)$, the set of solutions such that:

(1) all tasks are scheduled (possibly with $p_j = 0$) and, on each machine, tasks are scheduled in non-decreasing order of their deadlines without idle time;

(2) for all pairs of singular tasks T_{j_1}, T_{j_2} scheduled on the same machine, there exists at least one task T_k scheduled on the same machine between T_{j_1} and T_{j_2} and ending at its deadline

is dominant.
Furthermore, there exists a polynomial time algorithm that, given any optimal solution, returns an optimal solution satisfying those properties.

Proof. (1) has already been proven in Lemma I.3.1.
Let S be an optimal solution such that all tasks are scheduled without idle time and in non-decreasing order of their deadlines, and which is "minimal", in the sense that it has the minimum number x of pairs of singular tasks scheduled on the same machine without any task ending at its deadline in between. If $x = 0$ then the Lemma is proven.
Otherwise, let T_{j_1}, T_{j_2} be such a consecutive pair. Let k_1, k_2 be such that $p_{j_1}^{k_1} \leq p_{j_1} < p_{j_1}^{k_1+1}$ and $p_{j_2}^{k_2} \leq p_{j_2} < p_{j_2}^{k_2+1}$, and let k_1^ρ, k_2^ρ be such that $\rho_{j_1}^{k_1^\rho} < p_{j_1} < \rho_{j_1}^{k_1^\rho+1}$ and $\rho_{j_2}^{k_2^\rho} < p_{j_2} < \rho_{j_2}^{k_2^\rho+1}$. Remark that:

(1) There may be many processing times ($p_{j_1}^{k_1-1}$, $p_{j_1}^{k_1-2}\dots$) between $\rho_{j_1}^{k_1^\rho}$ and $p_{j_1}^{k_1}$, but they are all not interesting processing times, meaning null growth rate and no discontinuity. As a consequence, decreasing p_{j_1} to any value in $[\rho_{j_1}^{k_1^\rho}, p_{j_1}^{k_1}]$ yields a loss of profit of only $b_{j_1}^{k_1}(p_{j_1} - p_{j_1}^{k_1})$.

(2) If $b_{j_2}^{k_2} > 0$ then $p_{j_2}^{k_2+1}$ is an interesting processing time. As a consequence, increasing p_{j_2} to the next interesting processing time yields a gain of profit of $b_{j_2}^{k_2}(\rho_{j_2}^{k_2^\rho+1} - p_{j_2})$.

If $b_{j_1}^{k_1} = 0$, then we can set $p_{j_1} = \rho_{j_1}^{k_1^\rho}$ and shift all the jobs scheduled after to the left. The solution value is not downgraded (remark (1)) and x stricly decreases. Therefore, S was not minimal. The case $b_{j_2}^{k_2} = 0$ is similar.
If $0 < b_{j_1}^{k_1} \leq b_{j_2}^{k_2}$, we can reduce the processing time of T_{j_1} by $\min\{p_{j_1} - \rho_{j_1}^{k_1^\rho}, \rho_{j_2}^{k_2^\rho+1} - p_{j_2}\}$, shift all the regular tasks scheduled between T_{j_1} and T_{j_2} toward T_j and increase the processing time of T_{j_2} by the same quantity. In the process the profit is not downgraded (remarks (1) and (2), with $b_{j_1}^{k_1} \leq b_{j_2}^{k_2}$) and thus the solution remains optimal, but either T_{j_1} or T_{j_2} is now a regular task, and x strictly decreases. Therefore, S was not minimal.
If $b_{j_1}^{k_1} > b_{j_2}^{k_2} > 0$, we can increase the processing time of T_{j_1} by $\min\{\rho_{j_1}^{k_1^\rho+1} - p_{j_1}, \{d_l - C_l\}_{s_{j_1} \leq s_l < s_{j_2}}, p_{j_2} - \rho_{j_2}^{k_2^\rho}\}$ (with l restricted to tasks on the same machine as T_{j_1} and T_{j_2}), shift all the tasks scheduled between T_{j_1} and T_{j_2} toward T_{j_2} and decrease the processing time of T_{j_2} by that quantity; this is valid and the solution remains optimal, but either T_{j_1} is now a regular task, or T_{j_2} is now a regular task, or there exists a task T_l such that $C_l = d_l$ between T_{j_1} and T_{j_2}: in any case, x stricly decreases and therefore S was not minimal.
In all cases, there is a contradiction, so $x = 0$.
Furthermore, the operations described above can be used to strictly decrease x in an optimal solution with $x > 0$. Then, after a polynomial number of polynomial iterations, we can tranform any optimal solution in an optimal solution belonging to the dominant set. ■

A direct corollary from Lemma I.3.3 is that when all tasks have the same deadline, there will be at most one singular task on each machine:

Corollary I.3.1
For $P \mid \text{PLP}, d_j{=}d \mid -\sum w_j(p_j)$, the set of solutions such that:

(1) all tasks are scheduled (possibly with $p_j = 0$) and, on each machine, tasks are scheduled without idle time;

(2) on each machine, there is at most one singular task

is dominant.
Furthermore, there exists a polynomial time algorithm that, given any optimal solution, returns an optimal solution satisfying these properties.

I.4 NP-complete cases

Intuitively, some problems are NP-complete since several profit functions can be shaped to fit with the profit function of classical scheduling problems (for example $LPSTIP$, $b_j{=}0$). Then, reductions from $P \mid\mid C_{\max}$, $Pm \mid\mid C_{\max}$ or the Knapsack Problem will be straightforward. In the former case, the problems will be proven to be strongly NP-complete. In the latter cases, with fixed m, the problems will be only weakly NP-complete as we already know a pseudo-polynomial algorithm to solve them (Theorem I.3.1).
For one machine problems, we have:

Theorem I.4.1
$1 \mid d_j{=}d \mid -\sum w_j(p_j)$ is weakly NP-complete with the following profit functions:

(1) LPSTIP, $b_j{=}b$

(2) LBPST, $b_j{=}b$

(3) LBPSTIP, $b_j{=}b$, $w_j^{\max}{=}w^{\max}$

(4) LBPSTIP, $p_j^{\max}{=}p^{\max}$, $w_j^{\max}{=}w^{\max}$

Proof. As remarked before, those problems clearly belong to NP. Furthermore, they have already been proven to be solvable in pseudo-polynomial time (Theorem I.3.1). We prove their NP-completeness using reductions

from KNAPSACK, known to be NP-complete (Garey and Johnson 1979) and defined as follows.
Let I be an instance of KNAPSACK: given a finite set A of items and for each item $j \in A$ a size $a_j \in \mathbf{N}$ and a value $v_j \in \mathbf{N}$, and given bounds $B \in \mathbf{N}$ and $V \in \mathbf{N}$; the question is: is there a subset $S \subset A$ such that $\sum_{j \in S} a_j \leq B$ and $\sum_{j \in S} v_j \geq V$?

(1): KNAPSACK happens to be the particular case with $b_j = 0$: there are as many tasks as there are items, and for task T_j we set $w_j^{\min} = v_j$, $p_j^{\min} = a_j$ and we use B as the common due-date d.

(2) the idea is to set the growth rate to 1 and to stretch the minimum processing times and common deadline. The maximum processing time of a task is then relatively close to its minimum processing time, and every task scheduled, will necessarily be executed during its maximum processing time. Technically, let $v = n(\max_k v_k + 1)$ and let I' be an instance of $1 \mid \text{LBPST}, b_j{=}b, d_j{=}d \mid -\sum w_j(p_j)$ with n tasks T_j, $j = 1, \ldots, n$, such that $w_j^{\max} = v_j$, $b = 1$, $p_j^{\max} = va_j$, $p_j^{\min} = va_j - v_j$ (note that $p_j^{\min} \geq 0$) and $d = vB$; the question is: is there a schedule S such that $w(S) \geq V$?
If I has a "yes" answer, then clearly, I' has a "yes" answer too. If I' has a "yes" answer, then let S be an optimal solution of I'. Corollary I.3.1 provides a polynomial time algorithm to transform S such that it contains at most one task that is not scheduled during its maximum processing time (all tasks have the same deadline). Suppose that there is one such task T_j. The schedule is full (otherwise p_j could be increased and the solution value improved). Thus, $p(S) = \sum_{T_k \in S} p_k = d = vB$. Furthermore, $p(S) = \sum_{T_k \in S} p_k = v \sum_{T_k \in S} a_k - (p_j^{\max} - p_j)$. However $p_j^{\max} - p_j^{\min} < v$, therefore, $p(S)$ is not a multiple of v, which contradicts $p(S) = vB$. As a consequence, all tasks in S are scheduled during their maximum processing time and it is clearly possible to build the corresponding solution for KNAPSACK. Therefore, I has a "yes" answer.

For LBPSTIP, one could use the fact that LBPSTIP generalizes LPSTIP: hence a consequence of the first case (LPSTIP, $b_j = b$) is that LBPSTIP is NP-complete even if $b_j = b$. However, the proof relies on $b_j = 0$ and thus one must add the additional requirement that $w_j^{\min} = w_j^{\max}$; with such restrictions, LBPSTIP and LPSTIP are indeed the same problem and so there is nothing new. The two particular cases of LBPSTIP considered are different; we nevertheless use the same general idea.

(3): we set growth rates to 1 and the minimum profits large enough so that processing a task beyond its minimum processing time always yields a negligible profit, mimicking a classical scheduling problem.

Let I be an instance of KNAPSACK. Let I' be an instance of $1 \mid \text{LBPSTIP}, b_j{=}b,\ w_j^{\max}{=}w^{\max},\ d_j{=}d \mid -\sum w_j(p_j)$ with n tasks T_j, $j = 1, \ldots, n$, such that $b = 1$, $p_j^{\min} = a_j$, $w_j^{\min} = (B+1)v_j$, $w^{\max} = (B+1)(\max_k v_k + 1)$, $p_j^{\max} = a_j + (B+1)(\max_k v_k + 1 - v_j)$ and $d = B$; the question is: is there a schedule S such that $w(S) \geq (B+1)V$?
If I has a "yes" answer, then clearly, I' has a "yes" answer too. If I' has a "yes" answer, then let S be an optimal solution for I'. Corollary I.3.1 provides a polynomial algorithm to transform S such that it contains at most one task that is not scheduled during its minimum processing time in polynomial time (no task can be scheduled during its maximum processing time since they are all greater than the deadline). If there is no such task, then S directly provides an optimal solution to I. Otherwise, let T_j be this task. Let S' be a solution identical to S except that T_j is only scheduled during $p_j^{\min}$. Then $w(S') = \sum_{T_k \in S'} (B+1)v_k$ and it is therefore a multiple of $B+1$. However, $w(S) \geq (B+1)V$ and $w(S') - w(S) \leq d < B+1$. Therefore, $w(S') \geq (B+1)V$, and it is possible to build the corresponding solution for KNAPSACK. Therefore, I has a "yes" answer.

(4): we set the maximum processing time large enough so that processing a task beyond its minimum processing time always yield a negligible profit, mimicking a classical scheduling problem.
Let I' be an instance of $1 \mid \text{LBPSTIP}, p_j^{\max}{=}p^{\max},\ w_j^{\max}{=}w^{\max},\ d_j{=}d \mid -\sum w_j(p_j)$ with n tasks T_j, $j = 1, \ldots, n$, such that $p_j^{\min} = a_j$, $p^{\max} = B(\max_k v_k + 2)$, $x = \prod_{j=1}^{n}(B(\max_k v_k + 2) - a_j)$, $b_j = x\frac{\max v_k + 1 - v_j}{B(\max_k v_k + 2) - a_j}$, $w_j^{\min} = xv_j$, $w^{\max} = x(\max_k v_k + 1)$ and $d = B$; the question is: is there a schedule S such that $w(S) \geq xV$?
Note that all parameters are integers and all maximum profits are equal. If I has a "yes" answer, then clearly, I' has a "yes" answer too. If I' has a "yes" answer, then let S be an optimal solution of I'. Corollary I.3.1 provides a polynomial algorithm to transform S such that it contains at most one task that is not scheduled during its minimum processing time in polynomial time (no task can be scheduled during its maximum processing time since they are all greater than the deadline). If there is no such task, then S directly provides an optimal solution to I. Otherwise Let T_j be this task. Let S' be a solution identical to S except that T_j is only scheduled during $p_j^{\min}$. Then $w(S') = \sum_{T_k \in S'} xv_k$ and it is therefore a multiple of x. However, $w(S) \geq xV$ and $w(S') - w(S) \leq b_j d < x$. Therefore, $w(S') \geq xV$, and it is possible to build the corresponding solution for KNAPSACK. Therefore, I has a "yes" answer. ■

Note that the proofs for the LBPSTIP cases do not hold if, for all T_j, we

impose $p_j^{\max} \leq d_j$.

Now using a reduction from $P \mid\mid C_{\max}$ and $Pm \mid\mid C_{\max}$, we have:

Theorem I.4.2
$P \mid d_j{=}d \mid -\sum w_j(p_j)$ is strongly NP-complete and $Pm \mid d_j{=}d \mid -\sum w_j(p_j)$ is weakly NP-complete with the following profit functions:

(1) LPSTIP, $w_j^{\min}{=}w^{\min}$, $b_j{=}b$

(2) LBP, $w_j^{\max}{=}w^{\max}$

(3) LBP, $b_j{=}b$

(4) LBPST, $b_j{=}b$, $w_j^{\max}{=}w^{\max}$

(5) LBPSTIP, $p_j^{\min}{=}p^{\min}$, $b_j{=}b$, $w_j^{\max}{=}w^{\max}$

Proof. As remarked before, these problems clearly belong to NP. Furthermore, the cases with a fixed number of machines have already been proven solvable in pseudo-polynomial time (Theorem I.3.1 as special cases of PLP). We prove their NP-completeness using reductions from $Pm \mid\mid C_{\max}$ and $P \mid\mid C_{\max}$, respectively known to be NP-complete (Garey and Johnson 1979) and strongly NP-complete (Garey and Johnson 1978).
Let I be an instance of $Pm \mid\mid C_{\max}$: n tasks T_j, $j = 1, \ldots, n$, processing times $x_j \in \mathbf{N}$, $j = 1, \ldots, n$ and a deadline $\delta \in \mathbf{N}$; the question is: is there a m-processor schedule, $m \in \mathbf{N}$, for $\{T_j\}_{j=1,\ldots,n}$ that meets the overall deadline δ?

(1): $Pm \mid\mid C_{\max}$ happens to be a particular case of $Pm \mid$ LPSTIP, $w_j^{\min}{=}w^{\min}$, $b_j{=}b$, $d_j = d \mid -\sum w_j(p_j)$ with $b_j = 0$: there are as many tasks, and for task T_j we set $w^{\min} = 1$, $p_j^{\min} = x_j$ and $d = \delta$.

(2): let I' be an instance of $Pm \mid$ LBP, $w_j^{\max}{=}w^{\max}$, $d_j{=}d \mid -\sum w_j(p_j)$, with n tasks T_j, $j = 1, \ldots, n$, such that $p_j^{\max} = x_j$, $w^{\max} = \prod_{k=1}^{n} x_k$ and $d = \delta$; the question is: is there a schedule S such that $w(S) \geq n \prod_{k=1}^{n} x_k$? Note that for all $j = 1, \ldots, n$, $b_j = \frac{w^{\max}}{p_j^{\max}} = \prod_{k=1, k\neq j}^{n} x_k$ is an integer.
I responds "yes" if and only if each task T_j is scheduled during x_j before δ, yielding a solution to I'. Conversely, I' responds "yes" if and only if each task is scheduled during its maximum processing time, yielding a solution to I.

(3): we choose $d = \delta$ and $b = 1$ and $w_j^{\max} = x_j$ and thus $p_j^{\max} = x_j$; the question is: is there a schedule S such that $w(S) \geq \sum_{j=1}^{n} x_j$? The reasonning is the same.

(4): let I' be an instance of $1 \mid \text{LBPST}, b_j{=}b, w_j^{\max}{=}w^{\max}, d_j{=}d \mid -\sum w_j(p_j)$ with n tasks T_j, $j = 1, \ldots, n$, such that $w^{\max} = 1$, $b = 1$, $p_j^{\max} = x_j$, $p_j^{\min} = x_j - 1$ and $d = \delta$; the question is: is there a schedule S such that $w(S) \geq n$?
I responds "yes" if and only if each task T_j is scheduled during x_j before δ, yielding a solution to I'. Conversely, I' responds "yes" if and only if each task is scheduled during its maximum processing time, yielding a solution to I.

(5): let I' be an instance of $1 \mid \text{LBPSTIP}, p_j^{\min}{=}p^{\min}, b_j{=}b, w_j^{\max}{=}w^{\max}, d_j{=}d \mid -\sum w_j(p_j)$ with n tasks T_j, $j = 1, \ldots, n$, such that $p^{\min}{=}1$, $p_j^{\max} = x_j$, $b = 1$, $w^{\max} = \max_j x_j + 1$, $w_j^{\min} = w^{\max} + 1 - p_j^{\max}$ and $d = \delta$; the question is: is there a schedule S such that $w(S) \geq nw^{\max}$? Note that $w_j^{\min} \geq 0$.
I responds "yes" if and only if each task T_j is scheduled during x_j before δ, yielding a solution to I'. Conversely, I' responds "yes" if and only if each task is scheduled during its maximum processing time, yielding a solution to I.

In all cases, the transformations are not only polynomial but even strongly polynomial; hence the strong NP-completeness of the concerned cases. ■

Corollary I.4.1
$P \mid \text{PLP} \mid -\sum w_j(p_j)$ is strongly NP-complete. $Pm \mid \text{PLP} \mid -\sum w_j(p_j)$ is weakly NP-complete.

I.5 Polynomial cases solved with dynamic programming

In this section, we propose three new algorithms for PDPSP with a PLP function, that return optimal solutions in polynomial time for several profit functions.

I.5.1 $Pm \mid \textbf{PLP}, |\{p_j^k\}| \leq \kappa \mid -\sum w_j(p_j)$

The first algorithm is an adaptation of Algorithm 1 where instead of considering all instants $\{0, \ldots, d\}$, we only consider a polynomial subset of relevant

instants, thus reducing the number of states to a polynomial number.
From Lemma I.3.3, we can deduce the dominant set of possible start dates and end dates (we assume the tasks are sorted in non-decreasing order of their deadlines):

$$\Theta_0 = \left\{ d_j + \sum_{l=j+1}^{n} \rho_l \right\}_{\substack{j=0,\dots,n-1 \\ \rho_{j+1}\in\mathcal{P}_{j+1} \\ \vdots \\ \rho_n\in\mathcal{P}_n}} \cup \left\{ d_j - \sum_{l=1}^{j} \rho_l \right\}_{\substack{j=1,\dots,n \\ \rho_1\in\mathcal{P}_1 \\ \vdots \\ \rho_j\in\mathcal{P}_j}}$$

The size of this set is *a priori* exponential. However, it can be rewritten as follows. Let $\mathcal{P} = \cup_{j=1,\dots,n}\mathcal{P}_j$ and let

$$\Theta = \left\{ d_j + \sum_{\rho\in\mathcal{P}} l_\rho \rho \right\}_{\substack{j=0,\dots,n-1 \\ l_\rho=0,\dots,n-j \\ \sum l_\rho \le n-j}} \cup \left\{ d_j - \sum_{\rho\in\mathcal{P}} l_\rho \rho \right\}_{\substack{j=1,\dots,n \\ l_\rho=0,\dots,j \\ \sum l_\rho \le j}}$$

Clearly: $\Theta_0 \subset \Theta$, and

$$|\Theta| = O\left(n^{|\mathcal{P}|+1}\right)$$

In the general case, $|\mathcal{P}| = O(nK)$, but if $|\mathcal{P}|$ is independent of the instance size, $|\Theta|$ is polynomial in the instance size. For example, for LBPSTIP, $p_j^{\min}$=$p^{\min}$, $p_j^{\max}$=$p^{\max}$: $|\mathcal{P}| = 2$. More generally, we note $|\{p_j^k\}| \le \kappa$ to specify problems for which the number of different values for p_j^k is bounded by a integer κ independent of the input. Let us also remind that Pm in the first field of the three fields notation implies that m is also independent of the input.
Then we can adapt Algorithm 1 to run on instants of Θ instead of $\{0,\dots,d\}$:

Theorem I.5.1.1
Algorithm 2 returns an optimal solution to $Pm \mid \text{PLP}, |\{p_j^k\}| \le \kappa \mid -\sum w_j(p_j)$ in polynomial time.

Proof. The proof is similar to the one of Theorem I.3.1 in combination with Lemma I.3.3 which ensures that the problem is solved optimally.
The complexity of step 1 is $O(n \log n)$. The size of f^* is $n|\Theta|^m$, that is $O(n^{m(|\mathcal{P}|+1)+1})$, and since $O(m \log |\Theta|)$ is required to retrieve a value of f^*, it takes $O(mn^{|\mathcal{P}|+1}(m|\mathcal{P}| \log n + \log K)$ to compute a given value in step 2. The overall complexity is thus polynomial in the size of the input. ■

The complexity hierarchy among models allows to derive the following particular cases:

Algorithm 2 $Pm \mid \text{PLP} \mid -\sum w_j(p_j)$

INPUTS: an instance I

1: Sort tasks in non-decreasing order of their deadlines
2: Let $\mathcal{C} = \{C \in \Theta^m, C_1 \leq \cdots \leq C_m\}$. Return $\max_{C \in \mathcal{C}} f^*(n, C)$ such that:

$$f^*(j,C) = \begin{cases} 0 & \forall j = 0,\ldots,n \text{ and } C = \{0,\ldots,0\} \\ -\infty & \text{if } j = 0 \text{ and } C \neq \{0,\ldots,0\} \\ & \textit{(idle time at the beginning is never necessary)} \\ -\infty & \text{if } d_j < \max C_i \\ & \textit{(no task can end at } C_i\textit{)} \\ \max\limits_{\substack{1 \leq i \leq m \\ s \in \Theta, s \leq C_i}} f^*(j-1, C-(C_i-s)e^i) + w_j(C_i - s) & \text{otherwise} \end{cases}$$

where C is a m-dimensional vector containing the required completion time of the last task scheduled on each machine and e^i is the vector such that $e^i_i = 1$ and $e^i_k = 0$ for all $k \neq i$.

Corollary I.5.1.1

$Pm \mid \mid -\sum w_j(p_j)$ can be solved in polynomial time with the following profit functions:

- LPST (and thus LP)
- LBPST, $|\{p_j^{\max}\}| \leq \kappa$ (and thus LBP, $|\{p_j^{\max}\}| \leq \kappa$)
- LBPSTIP, $|\{p_j^{\min}, p_j^{\max}\}| \leq \kappa$ (and thus LPSTIP, $|\{p_j^{\min}\}| \leq \kappa$)

Besides, for PLP, if the pieces are all of the same duration (or more precisely, if there exists ρ_0 such that for all $\rho \in \mathcal{P}$, there exists $k \leq K$: $\rho = k\rho_0$), then:

$$\Theta' = \{d_j + l\rho_0\}_{\substack{j=0,\ldots,n \\ l=-jK,\ldots,(n-j)K}}$$

is a dominant set of instants of size

$$|\Theta'| = O\left(n^2 K\right)$$

Furthermore, a value $f^*(j, C)$ can be retrieved in $O(1)$ and $w_j(p)$ can be computed in $O(1)$. Therefore the complexity of the algorithm is dramatically reduced to $O(mn^{2m+3}K^{m+1} + n \log n)$.

I.5.2 $1 \mid \textbf{PLP}, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b\} \mid -\sum w_j(p_j)$

Algorithm 2 is handy when there is some regularity among the p_j^k, as that allows to enumerate all possible end dates. When there is no such regularity, the problem remains polynomial in some cases, provided some regularity on the profit. Again, the trick is to use dynamic programming schemes, but now the recursive functions do not represent a maximum profit for given completion times, but required time to achieve a given profit.

We will exhibit a polynomial algorithm for $1 \mid \text{PLP}, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b\} \mid -\sum w_j(p_j)$. First, consider the following dominant set:

Lemma I.5.2.1

For $1 \mid \text{PLP}, b_j^k \in \{0, b\} \mid -\sum w_j(p_j)$, the set of solutions $\mathcal{U}_1$ such that:

(1) all tasks are scheduled (possibly with $p_j = 0$) and on each machine, tasks are scheduled in non-decreasing order of their deadlines without idle time;

(2) there exists at most one singular task

is dominant.

Proof. (1) has already been proven in Lemma I.3.1.

Let S be an optimal solution such that all tasks are scheduled without idle time and in non-decreasing order of their deadlines, and which is "minimal", in the sense that it has the minimum number x of singular tasks. If $x \leq 1$ then the Lemma is proven.

Otherwise, let T_{j_1}, T_{j_2} be two singular tasks. Let k_1, k_2 be such that $p_{j_1}^{k_1} \leq p_{j_1} < p_{j_1}^{k_1+1}$ and $p_{j_2}^{k_2} \leq p_{j_2} < p_{j_2}^{k_2+1}$, and let k_1^ρ, k_2^ρ such that $\rho_{j_1}^{k_1^\rho} < p_{j_1} < \rho_{j_1}^{k_1^\rho+1}$ and $\rho_{j_2}^{k_2^\rho} < p_{j_2} < \rho_{j_2}^{k_2^\rho+1}$.

If $b_{j_1}^{k_1} = 0$, then we can set $p_{j_1} = \rho_{j_1}^{k_1^\rho}$ and shift all the tasks scheduled after to the left. The solution value is not downgraded and x stricly decreases. Therefore, S was not minimal. The case $b_{j_2}^{k_2} = 0$ is similar.

If $0 < b_{j_1}^{k_1} = b_{j_2}^{k_2}$, we can reduce the processing time of T_{j_1} by $\min\{p_{j_1} - \rho_{j_1}^{k_1^\rho}, \rho_{j_2}^{k_2^\rho+1} - p_{j_2}\}$, shift all the regular tasks scheduled between T_{j_1} and T_{j_2} toward T_{j_1} and increase the processing time of T_{j_2} by the same quantity. In the process the profit is not downgraded and thus the solution remains optimal, but either T_{j_1} or T_{j_2} is now a regular task, and x strictly decreases. Therefore, S was not minimal. ■

Even though dominant, $\mathcal{U}_1$ remains too large for an exhaustive search. We restrict it to another dominant set of smaller size.
Among the solutions with exactly one singular task T_l such that the tasks scheduled before T_l yield a profit w_C and the tasks scheduled after a profit w_S, we will only consider solutions such that T_l starts as early as possible and ends as late as possible. Therefore, we define the function $C^*(j,w)$ (resp. $s^*(j,w)$) that computes the earliest makespan (resp. the latest possible start date) to schedule tasks $T_1,\dots,T_j$ (resp. $T_j,\dots,T_n$) in this order with a profit w and such that all tasks are regular:

$$C^*(j,w) = \begin{cases} 0 & \text{if } w = 0 \\ +\infty & \text{if } j = 0 \text{ and } w \neq 0 \\ \min\limits_{\substack{k=1,\dots,K \\ w_j^k \leq w \\ C^*(j-1,w-w_j^k)+p_j^k \\ \leq d_j}} C^*(j-1, w-w_j^k) + p_j^k & \text{otherwise} \end{cases}$$

$$s^*(j,w) = \begin{cases} d_n & \text{if } w = 0 \\ -\infty & \text{if } j = n+1 \text{ and } w \neq 0 \\ \max\limits_{\substack{k=1,\dots,K \\ w_j^k \leq w \\ s^*(j+1,w-w_j^k) \leq d_j}} s^*(j+1, w-w_j^k) - p_j^k & \text{otherwise} \end{cases}$$

Lemma I.5.2.2
For $1 \mid \text{PLP}, b_j^k \in \{0,b\} \mid -\sum w_j(p_j)$, the set of solutions $\mathcal{U}_2$ such that, for all $S \in \mathcal{U}_2$:

(1) $S \in \mathcal{U}_1$;

(2) if S contains a singular task T_l, then for all $S' \in \mathcal{U}_1$ such that

- $\sum_{j=1}^{l-1} w_j(p_j(S')) = \sum_{j=1}^{l-1} w_j(p_j(S))$
- $\sum_{j=l+1}^{n} w_j(p_j(S')) = \sum_{j=l+1}^{n} w_j(p_j(S))$

then $C_{l-1}(S) \leq C_{l-1}(S')$ and $s_{l+1}(S) \geq s_{l+1}(S')$

is dominant.

Proof. (1) has already been proven by Lemma I.5.2.1.
Let $S \in \mathcal{U}_1$ be an optimal solution. If S contains no singular task, then

$S \in \mathcal{U}_2$. If S contains a singular task T_l, then let

$$S_c = \underset{\substack{S' \in \mathcal{U}_1 \\ \sum_{j=1}^{l-1} w_j(p_j(S'))=\sum_{j=1}^{l-1} w_j(p_j(S)) \\ \sum_{j=l+1}^{n} w_j(p_j(S'))=\sum_{j=l+1}^{n} w_j(p_j(S))}}{\arg\min} C_{l-1}(S')$$

$$S_s = \underset{\substack{S' \in \mathcal{U}_1 \\ \sum_{j=1}^{l-1} w_j(p_j(S'))=\sum_{j=1}^{l-1} w_j(p_j(S)) \\ \sum_{j=l+1}^{n} w_j(p_j(S'))=\sum_{j=l+1}^{n} w_j(p_j(S))}}{\arg\max} s_{l-1}(S')$$

and let S'' be the solution of $\mathcal{U}_1$ such that:

$$p_j(S'') = \begin{cases} p_j(S_c) & \text{if } j < l \\ p_l(S) & \text{if } j = l \\ p_j(S_s) & \text{if } j > l \end{cases}$$

Since S, S_c, and S_s are feasible, S'' is also feasible. Furthermore, $w(S'') = w(S)$ and thus S'' is optimal. Eventually, by construction, $S'' \in \mathcal{U}_2$. ■

Let

$$\Omega_0 = \left\{ \sum_{j=1}^{n} w_j \right\}_{\begin{pmatrix} w_1 \in \left\{w_1^k\right\}_{k=1,\dots,K_1} \\ \vdots \\ w_n \in \left\{w_n^k\right\}_{k=1,\dots,K_n} \end{pmatrix}}$$

For all $j = 1, \dots, n$, for all $w \notin \Omega_0$: $C^*(j, w) = +\infty$ and $s^*(j, w) = -\infty$. Thus, we only need to compute $C^*(j, w)$ and $s^*(j, w)$ when $w \in \Omega_0$.
As for Θ_0, the size of this set is *a priori* exponential. However, it can be rewritten as follows. Let $\mathcal{W} = \cup_{\substack{j=1,\dots,n \\ k=1,\dots,K_j}} \left\{w_j^k\right\}$ and let

$$\Omega = \left\{ \sum_{w \in \mathcal{W}} l_w w \right\}_{l_w \in \{0,\dots,n\}}$$

Clearly, $\Omega_0 \subset \Omega$, and

$$|\Omega| = O\left(n^{|\mathcal{W}|}\right)$$

In the general case, $|\mathcal{W}| = O(nK)$, but if $|\mathcal{W}|$ is independent of the instance size, $|\Omega|$ is polynomial in the instance size. For example, for LBPSTIP, $w_j^{\min}=w^{\min}$, $w_j^{\max}=w^{\max}$: $|\mathcal{W}| = 2$. We can therefore deduce a polynomial algorithm for the problem when the number of w_j^k is bounded:

Theorem I.5.2.1
Algorithm 3 returns an optimal solution to $1 \mid \text{PLP}, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b\} \mid -\sum w_j(p_j)$ in polynomial time.

Algorithm 3 $1 \mid \text{PLP}, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b\} \mid -\sum w_j(p_j)$

INPUTS: an instance I

1: Sort tasks in non-decreasing order of their deadlines
2: Compute $C^*(j, w)$ and $s^*(j, w)$ for all $(j, w) \in (\{0, \ldots, n\} \times \Omega)$
3: **return**

$$\max \begin{cases} \max\limits_{\substack{w \in \Omega \\ C^*(n,w) < +\infty}} & w & \textit{(no singular task)} \\ \max\limits_{\substack{(l, w_C, w_s) \\ \in (\{1,\ldots,n\} \times \Omega \times \Omega), \\ C^*(l-1, w_C) \\ \leq s^*(l+1, w_s)}} & w_C + w_s + w_l(s^*(l+1, w_s) - C^*(l-1, w_C)) & \\ & & \textit{(one singular task } T_l\textit{)} \end{cases}$$

Proof. Algorithm 3 searches exhaustively the dominant set $\mathcal{U}_2$ and therefore returns an optimal solution.
The complexity of step 1 is $O(n \log n)$. The sizes of C^* (resp. s^*) is $n|\Omega| = n^{|\mathcal{W}|+1}$ and since $O(\log |\Omega|)$ time is required to retrieve an already computed value of C^* (resp. s^*), it takes $O(K|\mathcal{W}| \log n)$ to compute a given value. Therefore, the complexity of step 2 is $O(K|\mathcal{W}|n^{|\mathcal{W}|+1} \log n)$. The complexity of computing the first max in step 3 is $O(|\mathcal{W}|n^{|\mathcal{W}|} \log n)$ and the complexity of computing the second one is $O(n^{2|\mathcal{W}|+1}(|\mathcal{W}| \log n + \log K))$. The overall complexity is thus polynomial in the size of the input. ■

Corollary I.5.2.1
$1 \mid \mid -\sum w_j(p_j)$ can be solved by Algorithm 3 in polynomial time with the following profit functions:

- LPSTIP, $|\{w_j^{\min}\}| \leq \kappa$, $b_j = b$
- LBPST, $b_j = b$, $|\{w_j^{\max}\}| \leq \kappa$
- LBPSTIP, $|\{w_j^{\min}, w_j^{\max}\}| \leq \kappa$, $b_j = b$

One may remark that the dominance property from Lemma I.5.2.2 can be generalized to the case of parallel machines. However, this will not lead to a polynomial algorithm (unless P = NP) as was proved by Theorem I.4.2.

I.5.3 $1 \mid \textbf{PLP}, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b_j\}, d_j = d \mid -\sum w_j(p_j)$

We will now propose our last algorithm, based on similar ideas. Tasks are now allowed to have different growth rates (still $b_j^k = b_j$), but they should have a common deadline. Since all deadlines are equal, Corollary I.3.1 directly provides a dominant set $\mathcal{U}_3$ such that:

(1) all tasks are scheduled (possibly with $p_j = 0$) in non-decreasing order of their deadlines and without idle time;

(2) for all $S \in \mathcal{U}_3$, S contains at most one singular task.

Once again, this set is too large for an exhaustive search and we restrict this set to another dominant set of smaller size.
We now define $\bar{C}^*(j, w, l)$ that computes the earliest makespan to schedule tasks $T_1, \ldots, T_j$ but not T_l with a profit of w such that all tasks are regular:

$$\bar{C}^*(j,w,l) = \begin{cases} \bar{C}^*(j-1,w,l) & \text{if } j = l \\ 0 & \text{if } w = 0 \\ +\infty & \text{if } j = 0 \text{ and } w \neq 0 \\ \min\limits_{\substack{k=1,\ldots,K \\ w_j^k \leq w}} \bar{C}^*(j-1, w-w_j^k, l) + p_j^k & \text{otherwise} \end{cases}$$

Lemma I.5.3.1
For $1 \mid \text{PLP}, b_j^k \in \{0, b_j\}, d_j = d \mid -\sum w_j(p_j)$, the set of solutions $\mathcal{U}_4$ such that, for all $S \in \mathcal{U}_4$:

(1) $S \in \mathcal{U}_3$

(2) if S contains a singular task T_l, then for all $S' \in \mathcal{U}_3$ such that $\sum_{j=1, j\neq l}^n w_j(p_j(S')) = \sum_{j=1, j\neq l}^n w_j(p_j(S))$, then $\sum_{j=1, j\neq l}^n p_j(S) \leq \sum_{j=1, j\neq l}^n p_j(S')$

is dominant.

Proof. (1) has already been proven by Corollary I.3.1.

Let $S \in \mathcal{U}_3$ be an optimal solution. If S contains no singular task, then $S \in \mathcal{U}_4$. If S contains a singular task T_l, then let

$$S_C = \underset{\sum_{j=1,j\neq l}^{n} w_j(p_j(S'))=\sum_{j=1,j\neq l}^{n} w_j(p_j(S))}{\underset{S'\in\mathcal{U}_3}{\arg\min}} \sum_{j=1,j\neq l}^{n} p_j(S')$$

and let S'' be the solution of $\mathcal{U}_3$ such that:

$$p_j(S'') = \begin{cases} p_j(S_C) & \text{if } j \neq l \\ p_l(S) & \text{if } j = l \end{cases}$$

Since S is feasible, S'' is also feasible. Furthermore, $w(S'') = w(S)$ and thus S'' is optimal. Eventually, by construction, $S'' \in \mathcal{U}_4$. ■

We can therefore deduce a polynomial algorithm for the problem when the number of w_j^k is bounded:

Theorem I.5.3.1
Algorithm 4 returns the optimal value of $1 \mid \text{PLP}, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b_j\}, d_j = d \mid -\sum w_j(p_j)$ in polynomial time.

Algorithm 4 $1 \mid \text{PLP}, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b_j\}, d_j = d \mid -\sum w_j(p_j)$

INPUTS: an instance I

1: Compute $C^*(j, w)$ for all $(j, w) \in (\{0, \ldots, n\} \times \{1, \ldots, n\})$ and
$\bar{C}^*(j, w, l)$ for all $(j, w, l) \in (\{0, \ldots, n\} \times \Omega \times \{1, \ldots, n\})$

2: **return**

$$\max \begin{cases} \max\limits_{\substack{w\in\Omega \\ C^*(n,w)\leq d}} & w & \textit{(no singular task)} \\ \max\limits_{\substack{l=1,\ldots,n \\ w\in\Omega \\ \bar{C}^*(n,w,l)\leq d}} & w + w_l(d - \bar{C}^*(n, w, l)) & \textit{(one singular task } T_l\textit{)} \end{cases}$$

Proof. Algorithm 4 searches exhaustively the dominant set $\mathcal{U}_4$ and therefore returns an optimal solution.
The complexity of step 1 is dominated by the computation of $\bar{C}^*$. The size of $\bar{C}^*$ is $n^2|\Omega| = n^{|\mathcal{W}|+2}$ and since $O(\log |\Omega|)$ time is required to retrieve an already computed value of $\bar{C}^*$, it takes $O(K|\mathcal{W}| \log n)$ to compute a given value. Therefore, the complexity of step 1 is $O(K|\mathcal{W}|n^{|\mathcal{W}|+2} \log n)$. The

complexity of computing the first max in step 2 is $O(|W|n^{|\mathcal{W}|}\log n)$ and the complexity of computing the second one is $O(n^{|\mathcal{W}|+1}(|W|\log n+\log K))$. The overall complexity is thus polynomial in the size of the input. ■

Corollary I.5.3.1
$1 \mid d_j{=}d \mid -\sum w_j(p_j)$ can be solved by Algorithm 4 in polynomial time with the following profit functions:

- LPSTIP, $|\{w_j^{\min}\}|{\leq}\kappa$
- LBPST, $|\{w_j^{\max}\}|{\leq}\kappa$
- LBPSTIP, $|\{w_j^{\min}, w_j^{\max}\}|{\leq}\kappa$

One may remark that the dominance property from Lemma I.5.3.1 can be generalized to the case of parallel machines. However, this will not lead to a polynomial algorithm (unless $P = NP$) as was proved by Theorem I.4.2.

In this section, we proposed several algorithms to solve different flavors of PLP functions. Even though polynomial, the proposed complexities may appear discouraging. However they are tremendously reduced when the algorithms are applied to particular cases, e.g., those presented in the corollaries. For example, Algorithm 5 is directly derived from Algorithm 4 to solve $1 \mid$ LPSTIP, $w_j^{\min}{=}w^{\min}$, $d_j{=}d \mid -\sum w_j(p_j)$ in $O(n^2)$.

Algorithm 5 $1 \mid$ LPSTIP, $w_j^{\min}{=}w^{\min}$, $d_j{=}d \mid -\sum w_j(p_j)$

INPUTS: an instance I

```
Sort tasks in non-decreasing order of their minimum processing time
OPT ← 0
for l from 1 to n do
    w ← w^min, C ← p_l^min
    OPT ← max{OPT, w + b_l(d − C)}
    for j from 1 to n, j ≠ l do
        w ← w + w^min, C ← C + p_j^min
        if C > d then
            break
        OPT ← max{OPT, w + b_l(d − C)}
return OPT
```

I.6 Polynomial cases solved with list algorithms

In this section, we present polynomial list algorithms for several particular cases.

I.6.1 $P \mid \mathbf{LP} \mid -\sum w_j(p_j)$

On a single machine and without additional constraint, scheduling jobs in non-increasing order of their growth rates, from the previous scheduled job end date until their deadline (when possible, *i.e.* when the previous scheduled job end date is before the current job deadline), leads to an optimal solution that generalizes to parallel machines:

Theorem I.6.1.1
Algorithm 6 returns an optimal solution of $P \mid \text{LP} \mid -\sum w_j(p_j)$ in polynomial time $O(n \log n)$.

Algorithm 6 $P \mid \text{LP} \mid -\sum w_j(p_j)$

if $m \geq n$ **then**
  for j from 1 to n **do**
    Schedule T_j on machine j on $[0, d_j[$.
else
  Sort jobs in non-increasing order of their growth rates b_j.
  Let q be a priority queue initialized such that element i has cost 0 for $i = 1, \ldots, m$.
    ▷ q contains the current completion time on each machine.
  for j from 1 to n **do**
    Get element i with minimum cost C from q.
    if $d_j > C$ **then**
      Schedule T_j on machine i on $[C, d_j[$.
      Update the cost of i in q to d_j.

I.6.2 $1 \mid \mathbf{LBP} \mid -\sum w_j(p_j)$

We already showed that the case with parallel machines is NP-complete even with a common deadline and either identical growth rates b_j=b or identical maximum processing-times $p_j^{\max}$=$p^{\max}$ (see Theorem I.4.2).
On the other hand, with a single machine and common deadlines ($d_j = d$), this problem corresponds to the well-known fractionnal knapsack problem.

This problem has been first solved by Dantzig 1957: it suffices to fill the knapsack with the items with the largest ratio value over weight until the knapsack is full. A $O(n)$ implementation is even possible (Kellerer et al. 2004).
In this section, we prove that, with arbitrary deadlines, the problem can still be solved in polynomial time.
A first approach uses linear programming and is once again based on Lemma I.3.1. The whole procedure is summed-up in Algorithm 7. However, more efficient algorithms can be designed.

Algorithm 7 $1 \mid \text{LBP} \mid -\sum w_j(p_j)$

1: Sort jobs in non-decreasing order of their deadlines.
2: Solve the following linear program:

$$\begin{cases} \max & \sum_{j=1}^{n} b_j p_j & \\ s.t. & \sum_{k=1}^{j} p_k \leq d_j & \forall j = 1, \ldots, n \\ & p_j \leq p_j^{\max} & \forall j = 1, \ldots, n \end{cases}$$

Indeed, the problem can be seen as the linear relaxation of the well-known $1||\sum w_j U_j$ where it is allowed to schedule only fractions of jobs. $1||\sum w_j U_j$ is NP-complete and solvable in pseudo-polynomial time (Lawler 1976). For the linear relaxation, Potts and Van Wassenhove 1988 proposed an explicit algorithm that schedules jobs by non-increasing order of deadlines for an overall complexity of $O(n \log n)$.
The problem is also equivalent to the scheduling problem with controllable processing-times presented by Janiak and Kovalyov 1996; the problem is to schedule on one machine n jobs, the processing-times of which can be decreased by allocating a continuously divisible resource. Janiak and Kovalyov 1996 proposed a polynomial time algorithm that schedules jobs by non-decreasing order of their deadlines and that also runs in $O(n \log n)$ time. This algorithm can be adapted to $1 \mid \text{LBP} \mid -\sum w_j(p_j)$ as follows: schedule jobs in non-decreasing order of their deadlines; whenever a job T_j cannot be scheduled during $p_j^{\max}$, make room for it by reducing the processing time of less profitable jobs already scheduled. As it generalizes to broader cases, the precise algorithm is presented later in this chapter; the impatient reader will find it as Algorithm 10 by setting $p_j^{\min} = 0$ for all $j = 1, \ldots, n$ (this assumption insures that the returned solution is feasible). Note that such a procedure builds, at iteration k, the optimal solution of the instance that only contains the first k jobs.
Whatever algorithm one prefers, the conclusion is:

Theorem I.6.2.1
Problem $1 \mid \text{LBP} \mid -\sum w_j(p_j)$ can be solved in polynomial time $O(n \log n)$.

I.6.3 $1 \mid$ LBPST, $p_j^{\min}=p^{\min}$, $b_j=b \mid -\sum w_j(p_j)$

The problem $1 \mid \text{LBPST}, b_j=b \mid -\sum w_j(p_j)$ is already NP-complete (see Theorem I.4.2). In this section, we propose a polynomial algorithm for the case with identical minimum processing-times and identical growth rates. Another polynomial case with parallel machines, based on a completely different approach, is presented in Section I.7.3.
As in the previous section, the set of solutions such that every job is scheduled and in non-decreasing order of their deadlines, is dominant. Thus, the problem can be reformulated as:

$$\begin{array}{lll} \max & \sum_{j=1}^{n} \max\{0, b(p_j - p^{\min})\} & \\ s.t. & p_j \leq p_j^{\max}, & \forall j = 1, \ldots, n \\ & \sum_{k=1}^{j} p_k \leq d_j, & \forall j = 1, \ldots, n \end{array} \tag{I.1}$$

but, as the objective is not linear, that does not prove that the problem is polynomial.
We nevertheless use this dominance property and implement it as a similar scheme as that proposed in Janiak and Kovalyov 1996, to get Algorithm 8. The remaining of this section is dedicated to prove the correctness of this algorithm. Roughly speaking: schedule jobs in non-decreasing order of their deadlines; whenever a job T_j cannot be scheduled during $p_j^{\max}$, make room for it by reducing the processing time of jobs with the smallest non null processing time.
The proof of correctness of such a scheme is inspired by the proof proposed by Janiak and Kovalyov 1996 but requires major technical adjustments. It relies on reducing the instance by updating deadlines. After the update, jobs may not be sorted in non-decreasing order of their deadlines anymore, but jobs are not reordered. Therefore, we consider Problem $(\bar{P})$ defined as follows:

$$\begin{array}{lll} \max & \sum_{j=1}^{n} \max\{0, b(p_j - p^{\min})\} & \\ s.t. & p_j \leq p_j^{\max}, & \forall j = 1, \ldots, n \\ & \sum_{k=1}^{j} p_k \leq d_j \text{ or } p_j = 0 & \forall j = 1, \ldots, n \end{array} \tag{$\bar{P}$}$$

Problem $(\bar{P})$ is similar to Formulation (I.1). The input and output are the same. However, in Formulation (I.1), the order was deduced from a dominance property; whereas in Problem $(\bar{P})$, the order is part of the instance,

Algorithm 8 1 | LBPST, $p_j^{\min}=p^{\min}$, $b_j=b$ | $-\sum w_j(p_j)$

Sort jobs in non-decreasing order of their deadlines d_j.
$C \leftarrow 0$ ▷ Current completion time.
Let q be a priority queue
▷ q will contain jobs currently scheduled and with non-null processing time; the cost of a job in q will be its processing time in the current solution.
$p_j \leftarrow p_j^{\max}$ for all $j = 1, \ldots, n$
for k from 1 to n **do**
    $C \leftarrow C + p_k^{\max}$
    Add element k with cost $p_k^{\max}$ in q.
    while $C > d_k$ **do**
        Let j be the minimal cost element of q.
        if $p_j \leq C - d_k$ **then**
            $C \leftarrow C - p_j$
            $p_j \leftarrow 0$
            Remove element j from q.
        else
            $p_j \leftarrow p_j - (C - d_k)$
            $C \leftarrow d_k$
return $\{p_j\}_{j=1,\ldots,n}$

i.e. jobs cannot be reordered. Furthermore, jobs may not be sorted in non-decreasing order of deadlines in ($\bar{P}$). This has two consequences. First, a job may not be scheduled at all; this corresponds to $p_j = 0$ in Problem ($\bar{P}$). Second, if $p_j = p_j^{\max}$, the solution may not be feasible because of the deadlines of the following jobs. Hence the need to define ρ_j^k and ρ_j:

$$\rho_j^k = \min\left\{p_j^{\max}, \min_{u, j \leq u \leq k} d_u\right\}$$
$$\rho_j = \rho_j^n$$

ρ_j can be interpreted as the effective maximum processing time of job T_j in any feasible solution. ρ_j^k can be defined equivalently recursively:

$$\rho_j^0 = p_j^{\max}$$
$$\rho_j^k = \min\left\{\rho_j^{k-1}, d_k\right\}$$

From these definitions, one can derive several properties. If, for some $k < n$, there exists l_k such that $\rho_{l_k}^k \geq \rho_j^k$ for all $j = 1, \ldots, k$, then $\rho_{l_k}^{k+1} \geq \rho_j^{k+1}$. If $k_2 \geq k_1$, then $\rho_j^{k_2} \leq \rho_j^{k_1}$. If $j_2 \geq j_1$ and $\rho_{j_1}^k < p_{j_2}^{\max}$, then $\rho_{j_2}^k \leq \rho_{j_1}^k$.

Besides, remark that if I is an instance of $1 \mid \text{LBPST}, p_j^{\min}=p^{\min}, b_j=b \mid -\sum w_j(p_j)$ and $\bar{I}$ an instance of Problem ($\bar{P}$) with the same input provided such that d_j are sorted in non-decreasing order, then an optimal solution of $\bar{I}$ is clearly an optimal solution of I.
Now let us start with the following dominance property:

Lemma I.6.3.1
Let $\rho_{\max} = \max_{j=1..n} \rho_j$. For a given job T_l such that $\rho_l = \rho_{\max}$, there exists an optimal solution to ($\bar{P}$) in which $p_l = \rho_l$.

Algorithm 9

```
1:  procedure INCREASEBEST(\{p_j\}_{j=1,...,n}, l)
2:      p'_j ← p_j for all j = 1, ..., n
3:      if p'_l = ρ_max then
4:          return \{p'_j\}_{j=1,...,n}
5:      for k from 0 to n do
6:          if p_k = 0 or k = l then
7:              continue
8:          if p'_k < ρ_max − p'_l then
9:              p'_l ← p'_l + p'_k                          ▷ p'_l < ρ_max
10:             p'_k ← 0
11:         else
12:             p'_k ← p'_k + p'_l − ρ_max
13:             p'_l ← ρ_max
14:             return \{p'_j\}_{j=1,...,n}
```

Proof. Let S be an optimal solution. We prove that Algorithm 9 applied to S returns a solution S' which remains optimal and such that $p'_l = \rho_{\max}$.
First, note that the solution value is never degraded: $\sum_{k=0}^{n} p'_k$ is constant throughout the algorithm and the number of scheduled jobs is non-increasing. Second, the algorithm stops either at line 4 or line 14. Otherwise, after the for loop, $p'_k = 0$ for all $k \neq l$ and $p'_l < \rho_{\max}$, corresponding to a sub-optimal solution. As the solution value is never degraded that would contradict that S was optimal.
If the algorithm stops at line 4, then the initial solution S was as expected. Otherwise, it stops at the end of iteration k and $p'_j = 0$ for all $j \leq k$, $j \neq l$; therefore constraint j of $\bar{P}$ is satisfied for all $j \leq k$, $j \neq l$. Furthermore, for all $j > k$, $j \neq l$, $C'_j = C_j$; therefore, constraint j is also satisfied for $j > k$,

$j \neq l$. Finally, if $k \leq l$, $C'_l = C_l$, and if $k > l$, $p'_j = 0$ for all $j < l$; therefore constraint l is also satisfied. ■

Lemma I.6.3.2
Let I be an instance of Problem ($\bar{P}$). There exists a job T_l such that $\rho_l = \rho_{\max}$ and such that Algorithm 8 executed on I sets p_l at ρ_l.
(NOTE: Since jobs are not allowed to be reordered in Problem $\bar{P}$, the first step of Algorithm 8 is skipped, whenever applied to an instance of Problem $\bar{P}$.)

Proof. Let $\rho^k_{\max} = \max_{j=1,\dots,k} \rho^k_j$ be the maximum effective processing time at iteration k. We show that at each iteration k, there exists T_{l_k} such that $p^k_{l_k} = \rho^k_{\max}$. T_{l_k} is therefore necessarily at the end of the queue. In the case where there are several jobs at $\rho^k_{\max}$, we consider T_{l_k} to be the last one in the queue.
It is clearly true at the end of iteration 1.
Let us suppose the property true for some $k < n$. Then, at iteration $k+1$, job T_{k+1} is added to the queue with processing time $p^{\max}_{k+1}$ (in what follows, p^k_j is the processing time of job T_j at the end of iteration k):

- Case $p^{\max}_{k+1} < p^k_{l_k}$. Then $\rho^{k+1}_{k+1} < \rho^k_{\max}$. Therefore $\rho^{k+1}_{\max} = \rho^{k+1}_{l_k}$. T_{l_k} stays at the end of the queue and $p^{k+1}_{l_k} = \rho^{k+1}_{l_k}$.
- Case $p^{\max}_{k+1} \geq p^k_{l_k}$. Then $\rho^{k+1}_{k+1} \geq \rho^{k+1}_{l_k} \geq \rho^{k+1}_j$ for all $j = 1, \dots, k$. T_{k+1} takes the last position in the queue and $p^{k+1}_{k+1} = \rho^{k+1}_{k+1}$ and $\rho^{\max}_{k+1} = \rho^{k+1}_{k+1}$.

■

Let I be an instance of Problem ($\bar{P}$). From Lemma I.6.3.2, we know that there exists a job that Algorithm 8 sets at $\rho_{\max}$. Let T_l be such a job. We now define the following reduced instance I' (of Problem ($\bar{P}$)):

- $d'_k = d_k$ for all $k = 1, \dots, l-2$
- $d'_{l-1} = \min\{d_{l-1}, d_l - \rho^l_l\}$ (if $l > 1$)
- $d'_k = d_k - \rho^k_l$ for all $k = l+1, \dots, n$ (ensuring $d'_k \geq 0$)
- T_l is removed from I'

Note that the jobs may not be sorted in non-decreasing order of deadlines anymore.

Lemma I.6.3.3
Let I be an instance of $\bar{P}$ and I' (one of) its reduced instance. If S' is an optimal solution of I', then $S = \{p'_1, \dots, p'_{l-1}, \rho_l, p'_{l+1}, \dots, p'_n\}$ is an optimal solution of I.

Proof. First, we prove that S is feasible.
If $\rho_l = p_l^{\max}$: For $k < l-1$, $\sum_{j=1}^{k} p_j = \sum_{j=1}^{k} p'_j \leq d'_k = d_k$. For $k = l-1$, $\sum_{j=1}^{l-1} p_j = \sum_{j=1}^{l-1} p'_j \leq d'_{l-1} = d_l - p_l^{\max} \leq d_{l-1} + p_l^{\max} - p_l^{\max} = d_{l-1}$. For $k \geq l+1$, $\sum_{j=1}^{k} p_j = \sum_{j=1, j\neq l}^{k} p'_j + \rho_l \leq d'_k + p_l^{\max} = d_k - p_l^{\max} + p_l^{\max} = d_k$. Therefore, all constraints are satisfied.
If $\rho_l < p_l^{\max}$, then there exists a last $k^* \geq l$ such that $d_{k^*} = \rho_l = \rho_l^{k^*}$ and therefore $d'_{k^*} = 0$ and for all $j \leq k^*$, $j \neq l$: $p'_j = 0$. Therefore, constraints 1 to k^* are satisfied. Furthermore, for $k \geq k^*+1$: $\sum_{j=1}^{k} p_j = \sum_{j=k^*+1}^{k} p'_j + \rho_l \leq d'_k + \rho_l^{k^*} = d_k - \rho_l^k + \rho_l^{k^*} \leq d_k$. Therefore, all constraints are satisfied.
Now, let suppose that S is not optimal, *i.e.* there exists an optimal solution S^* of I such that $w(S^*) > w(S)$. From Lemma I.6.3.1, we can consider a S^* such that $p_l^* = \rho_l$. Let $S'^* = \{p_1^*, \ldots, p_{l-1}^*, p_{l+1}^*, \ldots, p_n^*\}$ be a solution of I'. $w(S'^*) = w(S^*) - w_l(\rho_l) > w(S) - w_l(\rho_l) = w(S')$. Therefore, if S'^* is feasible, it contradicts S' optimal.
Let us show that S'^* is feasible. For $k < l$: $\sum_{j=1}^{k} p'^*_j = \sum_{j=1}^{k} p_j^* \leq d_k = d'_k$. For $k > l$: $\sum_{j=1, j\neq l}^{k} p'^*_j = \sum_{j=1}^{k} p_j^* - \rho_l \leq d_k - \rho_l = d'_k + \rho_l^k - \rho_l \leq d'_k$. Therefore, all constraints are satisfied. ■

Lemma I.6.3.4
Let I be an instance of Problem ($\bar{P}$) and I' (one of) its reduced instance. Let S and S' be the solutions returned by Algorithm 8 respectively on instance I and I'. Then for all $k = 1, \ldots, n$, $k \neq l$: $p_k = p'_k$

Proof. In what follows, C^k denotes the completion time of the current solution at the end of iteration k.

- Case $\rho_l = p_l^{\max}$: let $\hat{k} = l$. Clearly, for all $k < l-1$, for all $j = 1, \ldots, k$: $p_j^k = p'^k_j$. At iteration $l-1$ for I, $C^{l-2} + p_{l-1}^{\max} - C^{l-1}$ units of processing time are removed from q.
  - if $C^{l-2} + p_{l-1} \leq d_{l-1}$ and $C^{l-1} + p_l^{\max} \leq d_l$, then no unit of processing time is removed from q, neither for I at iterations $l-1$ and l, nor for I' at iteration $l-1$.
  - if $C^{l-2} + p_{l-1} > d_{l-1}$ and $C^{l-1} + p_l^{\max} \leq d_l$: no other unit of processing time are removed from q at iteration l. Furthermore, $d'_{l-1} = d_{l-1}$ and therefore the same number of units of processing time is removed at iteration $l-1$ for I'.
  - if $C^{l-1} + p_l^{\max} > d_l$ (which implies $C_l = d_l$): $C^{l-1} + p_l^{\max} - C^l$ units of processing time are removed from q at iteration l on I. The total number of units of processing time removed from q at iteration l

and $l-1$ for I is therefore $C^{l-2}+p_{l-1}^{\max}+p_l^{\max}-d_l$. Furthermore $C'^{l-1}=d'_{l-1}=d_l-p_l^{\max}$ and therefore $C'^{l-2}+p_{l-1}^{\max}-C'^{l-1}=C^{l-2}+p_{l-1}^{\max}+p_l^{\max}-d_l$ units of processing time are removed at iteration $l-1$ for I'.

The same amount of processing time is removed for both instances and since by definition of T_l, p_l is not modified, both queue are identical after iteration l except for p_l.

- Case $\rho_l < p_l^{\max}$: let $\hat{k}$ be the first iteration such that $p_l^{\hat{k}}=\rho_l$. Then for all $j=1,\dots,\hat{k}$, $j\neq l$, $p_j^{\hat{k}}=0$. Furthermore $d'_{\hat{k}}=0$, and thus $p'_j{}^{\hat{k}}=0$.

In both cases, for the remaining iterations, Algorithm 8 clearly behaves the same on I and I' since: q and q' are the same (except for p_l); on I, by definition of T_l, p_l will not be modified; and on I', $d'_k=d_k-\rho_l$ for all $k=\hat{k}+1,\dots,n$. ■

We now can prove the correctness of Algorithm 8:

Theorem I.6.3.1
Algorithm 8 returns an optimal solution of problem $1\mid \text{LBPST}, p_j^{\min}{=}p^{\min}, b_j{=}b \mid -\sum w_j(p_j)$ in polynomial time $O(n\log n)$.

Proof. Algorithm 8 returns an optimal solution on any instance of Problem $(\bar{P})$ with 0 job. Let suppose that it returns an optimal solution on any instance of Problem $(\bar{P})$ with $n-1$ jobs, and let us show that it then returns an optimal solution on any instance of Problem $(\bar{P})$ with n jobs. Let I be an instance with n jobs and I' (one of) its reduced instance. Using the induction hypothesis, the solution S' returned by the algorithm on I' is optimal. From Lemma I.6.3.3 and Lemma I.6.3.4, the solution returned by the algorithm on I is also optimal. Therefore, the algorithm returns the optimal solution for any instance of Problem $(\bar{P})$, and consequently of Problem $1\mid \text{LBPST}, p_j^{\min}{=}p^{\min}, b_j{=}b \mid -\sum w_j(p_j)$. Furthermore, it runs in $O(n\log n)$. ■

I.6.4 $1\mid$ LBPSTIP $\mid -\sum w_j(p_j)$

This problem is very close to the scheduling problem with controllable processing times where the processing time is a linear decreasing function of the amount of a common resource allocated to the job. The difference is that, in our problem, we may not schedule a job. Therefore we adapt from Janiak and Kovalyov 1996 the following property:

Lemma I.6.4.1
Let V be a set of tasks such that there exists an optimal solution S which set of scheduled jobs is V, *i.e.* $V = \left\{T_j \in S, p_j(S) \geq p_j^{\min}\right\}$.
If we know such a set V, then Algorithm 10 returns an optimal solution to $1 \mid \text{LBPSTIP} \mid -\sum w_j(p_j)$ in polynomial time $O(n \log n)$.

Algorithm 10

```
1:  procedure SCHEDULE(V)
2:      Sort jobs in non-decreasing order of their deadlines d_j.
3:      C ← 0                                          ▷ Current completion time.
4:      for k from 1 to |V| do
5:          C ← p_k^min
6:          if C > d_k then
7:              return                                 ▷ No feasible solution.
8:      C ← 0
9:      Let q be a priority queue
10:         ▷ q will contain scheduled jobs sorted in non-decreasing order of
    their growth rates.
11:     p_j ← p_j^max for all j = 1, ..., |V|
12:     for k from 1 to |V| do
13:         C ← C + p_k^max
14:         Add element k with cost b_k in q.
15:         while C > d_k do
16:             Let j be the minimal cost element of q.
17:             if p_j − p_j^min ≤ C − d_k then
18:                 C ← C − (p_j − p_j^min)
19:                 p_j ← p_j^min
20:                 Remove element j from q.
21:             else
22:                 p_j ← p_j − (C − d_k)
23:                 C ← d_k
24:     return Σ_{j=1}^{|V|} b_j p_j
```

Unfortunately, in the general case, finding such a set V has been shown to be NP-complete. Therefore, we will look for special cases for which one can be found in polynomial time.
For a given instance, let $\mathcal{T}$ be the set of jobs and let $T_{j_1}, T_{j_2} \in \mathcal{T}$. We say that T_{j_1} dominates T_{j_2} if for any optimal solution S such that $T_{j_2} \in S$ and $T_{j_1} \notin S$, there exists an optimal solution S' containing all the tasks of S

except T_{j_2} and containing T_{j_1}. For example, if $p_{j_1}^{\min} \leq p_{j_2}^{\min}$ and $p_{j_1}^{\max} \leq p_{j_2}^{\max}$ and $w_{j_1}^{\min} \geq w_{j_2}^{\min}$ and $w_{j_1}^{\max} \geq w_{j_2}^{\max}$ and $d_{j_1} \leq d_{j_2}$ then T_{j_1} dominates T_{j_2}. This dominance defines a partial order of $\mathcal{T}$. Thus, in an optimal solution, there is no reason to schedule a task if another task that dominates it has not been scheduled either:

Lemma I.6.4.2
Let $\mathcal{U}_2$ be the set of solutions such that for all $S \in \mathcal{U}_2$, for all $T_{j_1} \in S$ and $T_{j_2} \notin S$, T_{j_1} does not dominate T_{j_2}. $\mathcal{U}_2$ is dominant.

In order to evaluate the size of this set, we show that any of its solutions can be characterized only by its "worst tasks":

Lemma I.6.4.3
For all $V \in \mathcal{U}_2$, let

$$\begin{aligned} f : \ & \mathcal{U}_2 \to \text{the set of antichains of } \mathcal{T} \\ & V \mapsto \{T_j, \quad \forall T_q \in V, q \neq j, \quad T_j \text{ does not dominate } T_q\} \end{aligned}$$

f is bijective.

Proof. Clearly, for all $V \in \mathcal{U}_2$, $f(V)$ is indeed an antichain of $\mathcal{T}$. Therefore, f is well defined.
First, we show that f is surjective: if A is an antichain of $\mathcal{T}$, $f^{-1}(A)$ is then obtained recursively by adding all tasks dominating tasks of A. Note that this computation can be achieved in polynomial time.
Now, we show that f is injective: let $V_1, V_2 \in \mathcal{U}_2$ such that $f(V_1) = f(V_2)$. Then

- every task of $f(V_1) = f(V_2)$ belongs to V_1 and V_2;
- therefore, by definition of $\mathcal{U}_2$, every task that dominates a job of $f(V_1) = f(V_2)$ belongs to V_1 and V_2;
- furthermore, every task outside $f(V_1) = f(V_2)$ that does not dominate a task of $f(V_1) = f(V_2)$ does not belong to V_1 nor V_2.

Therefore $V_1 = V_2$. ■

Theorem I.6.4.1
Algorithm 11 returns an optimal solution to $1 \mid \text{LBPSTIP} \mid -\sum w_j(p_j)$. If four parameters among $\{p_j^{\min}, p_j^{\max}, b_j, w_j^{\min}, w_j^{\max}, d_j\}$ can only take a fixed number of possible values, then Algorithm 11 returns an optimal

solution to $1 \mid \text{LBPSTIP} \mid -\sum w_j(p_j)$ in polynomial time.

Algorithm 11 $1 \mid \text{LBPSTIP} \mid -\sum w_j(p_j)$

1: **for** every antichain A of $\mathcal{T}$ **do**
2: $S \leftarrow \text{SCHEDULE}(f^{-1}(A))$ ▷ SCHEDULE is Algorithm 10
3: $\text{OPT} \leftarrow \max\{\text{OPT}, w(S)\}$
4: **return** OPT

Proof. Lemma I.6.4.1, I.6.4.2 and I.6.4.3 imply the correcteness of Algorithm 11.
If two jobs have four identical parameters among

$$\left\{p_j^{\min}, p_j^{\max}, b_j, w_j^{\min}, w_j^{\max}, d_j\right\}$$

then they are comparable, *i.e.* one dominates the other. Therefore, the number of antichains is bounded by n to the power of the product of the possible values for these fixed valued parameters. If the value of this product is bounded by a constant independent of the instance, then Algorithm 11 runs in polynomial time. ■

I.7 Polynomial cases solved with maximum weight b-matching

In this section, we exhibit polynomial algorithms for three processing time dependent profit maximization scheduling problems with parallel machines. Those algorithms use as subproblem the maximum weight b-matching which can be solved in polynomial time (Schrijver 2002).
Given a graph $G(V, E)$, a demand/supply b_v for each vertex $v \in V$, and a weight w_e for each edge $e \in E$, a b-matching of G is a vector $x \in \mathbf{N}^E$ such that $x(\delta_v) \leq b_v$ for all $v \in V$ (where δ_v is the set of edges incident to v). The weight of a b-matching $x \in \mathbf{N}^E$ is defined as $\sum_{e \in E} w_e x_e$. The maximum weight b-matching problem is then the problem of finding a b-matching of maximum weight in G.

I.7.1 $P \mid \textbf{LPSTIP}, p_j^{\min}{=}p^{\min}, d_j{=}d \mid -\sum w_j(p_j)$

The algorithm consists in solving a polynomial number of maximum weight b-matching problems which rely on the dominant set described below.

We consider the case $d = qp^{\min} + r$, with $q, r \in \mathbf{N}$, $q \geq 2$, $0 < r < p^{\min}$. However, with a similar reasoning, the results can be adapted for the case $d = qp^{\min}$, $q \in \mathbf{N}$, $q \geq 2$. The case $d < 2p^{\min}$ is trivial.

Lemma I.7.1.1
Let $\mathcal{U}$ be the set of solutions such that for all $S \in \mathcal{U}$:

- for all $T_j \in S$, $p_j \geq p^{\min}$;
- there exists at most one job $T_j \in S$ such that $p_j \notin \{d, p^{\min}, p^{\min} + r\}$.

$\mathcal{U}$ is dominant.

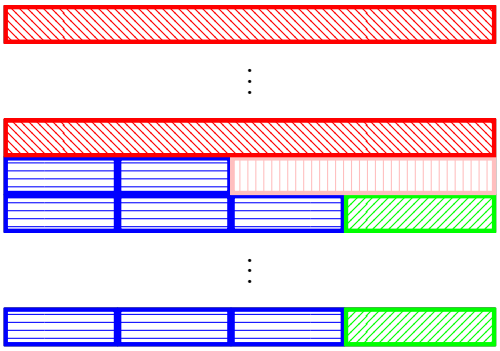


Figure I.8 – Illustration of the structure of the solutions in $\mathcal{U}$

Proof. Let T_j be a break job iff $p_j > p^{\min}$ and let T_j be a special job iff:

$$p_j > p^{\min} \text{ and } p_j \neq p^{\min} + r \text{ and } p_j \neq d$$

Figure I.8 illustrates the structure of the solutions of $\mathcal{U}$. All jobs except the ones with horizontal stripes are break jobs and the job with vertical stripes is a special job.
Thus, $\mathcal{U}$ is the set of solutions for which each scheduled job is at least executed during the minimum processing time and with at most one special job.
Note that a special job is also a break job, and if a job T_j is the only special job scheduled on a machine, then $p_j = tp^{\min} + r$, $t \in \mathbf{N}$, $t > 1$ and the other jobs scheduled on this machine have processing time $p^{\min}$.
Let $S \notin \mathcal{U}$ be an optimal solution. Let us build another solution $S' \in \mathcal{U}$ from S without degrading its value.
First, we can remove all jobs with an allocated execution time strictly shorter than $p^{\min}$ without degrading the solution value.

Then we ensure that for each machine, there is only one break job. If it is not the case, let T_j and T_k ($b_j \geq b_k$) be two break jobs scheduled on the same machine. Let's transfer $p_k - p^{\min}$ processing time from T_k to T_j. The value of the solution is not degraded and the number of break jobs on this machine strictly decreases. Thus, by repeating this operation, we get a solution as good as S with at most one break job per machine.
Finally, we ensure that there is only one special job in the solution. If it is not the case, let T_j and T_k ($b_j \geq b_k$) be two special jobs of S. From the previous step, we know that T_j and T_k are not scheduled on the same machine (let's call M_{i_j} and M_{i_k} the machine on which T_j and T_k are respectively scheduled), but they are the only special jobs on their machine. This implies that the other jobs scheduled on those machines all have processing time $p^{\min}$ and that $p_j = t_j p^{\min} + r$ and $p_k = t_k p^{\min} + r$, with $t_j, t_k \in \mathbf{N}$, $t_j, t_k > 1$. Now, while T_j and T_k remain special jobs, we move a job (different from T_j) from M_{i_j} to M_{i_k}, increase processing time of T_j by $p^{\min}$ and decrease processing time of T_k by $p^{\min}$. Thus, in a finite number of steps, we have either $p_j = d$ or $p_k = p^{\min} + r$ and the number of special jobs have strictly decreased. Therefore, we can repeat the operation till there remains only one special job in the solution. ■

Now we can prove the theorem:

Theorem I.7.1.1
There exists an algorithm that solves $P \mid \text{LPSTIP}, p_j^{\min}{=}p^{\min}, d_j{=}d \mid -\sum w_j(p_j)$ in polynomial time.

Proof. Following Lemma I.7.1.1, we only focus on solutions of $\mathcal{U}$. Thus, we can partition the jobs of a solution $S \in \mathcal{U}$ into 4 subsets depending on their processing time (and one more for the non scheduled jobs):

$$
\begin{aligned}
U_1(S) &= \{T_j \in S, \quad p_j = d\} \\
U_2(S) &= \{T_j \in S, \quad T_j \text{ is a special job}\} \\
U_3(S) &= \left\{T_j \in S, \quad p_j = p^{\min} + r\right\} \\
U_4(S) &= \left\{T_j \in S, \quad p_j = p^{\min}\right\} \\
U_0(S) &= \{T_j \notin S\}
\end{aligned}
$$

$$
\forall k \in \{0, \ldots, 4\}, \quad n_k(S) = |U_k(S)|
$$

$$
t(S) = \begin{cases} 0, & \text{if } U_2(S) = \emptyset \\ \frac{d - p_j}{p^{\min}}, & \text{if } U_2(S) = \{T_j\} \end{cases}
$$

$t(S)$ represents the number of jobs scheduled on the same machine as the job of U_2. Note that, if $U_2(S) = \{T_j\}$, then $p_j = d - t(S)p^{\min}$.
Furthermore, for all $S \in \mathcal{U}$

$$0 \leq n_1(S) \leq m \qquad 0 \leq t(S) \leq n-1$$

$$\begin{aligned}
n_2(S) &= \begin{cases} 0, & \text{if } t(S) = 0 \\ 1, & \text{otherwise} \end{cases} \\
n_3(S) &= m - n_1(S) - n_2(S) \\
n_4(S) &= (q-1)n_3(S) + t(S) \qquad \text{(remember that } q = \lfloor d/p^{\min} \rfloor) \\
n_0(S) &= n - n_1(S) - n_2(S) - n_3(S) - n_4(S)
\end{aligned}$$

Therefore, if $n_1(S)$ and $t(S)$ are fixed, we can determine $n_2(S)$, $n_3(S)$, $n_4(S)$ and $n_0(S)$. Thus, the optimal value is the best optimal value of the $O(n^2)$ problems with those parameters fixed:

$$\text{OPT} = \max_{(n_1,t)\in(\{0,\ldots,m\}\times\{0,\ldots,n-1\})} \text{OPT}(n_1, t)$$

Now, we show that for all $(n_1, t) \in \{1, \ldots, m\} \times \{0, \ldots, n-1\}$, a maximum weight b-matching problem model can compute $\text{OPT}(n_1, t)$. We create n nodes T_j, $j = 1, \ldots, n$ with supply 1, and a 5 nodes U_i, $i = 0, \ldots, 4$ with demand n_i such that $n_2 = 0$ if $t \leq 1$, 1 otherwise; $n_3 = m - n_1 - n_2$, $n_4 = (q-1)n_3 + t$, $n_0 = n - n_1 - n_2 - n_3 - n_4$. Then for all $j = 1, \ldots, n$, for all $i = 0, \ldots, 4$, we add the arc (T_j, U_i) with capacity $u_{ji} = 1$ and cost:

$$c_{ji} = \begin{cases} w_j(d) & i = 1 \\ w_j(d - tp^{\min}) & i = 2 \\ w_j(p^{\min} + r) & i = 3 \\ w_j^{\min} & i = 4 \\ 0 & i = 0 \end{cases}$$

A part of the graph including only one job is represented in Figure I.9. The size of the graph is polynomial compared to the size of the instance. Furthermore, the number of problems that we have to solve is $O(n^2)$. Therefore, the problem can be solved in polynomial time. ■

I.7.2 $P \mid \textbf{LPSTIP}, p_j^{\min}{=}p^{\min}, b_j{=}b \mid -\sum w_j(p_j)$

Compared to the previous case, we relax the property that all deadlines are equal, which is replaced by equal growth rates.

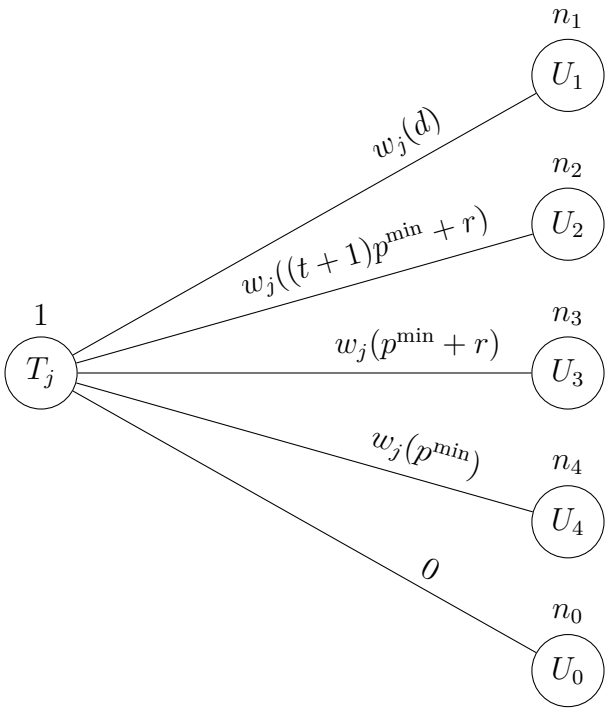


Figure I.9 – A part of the graph including only one job given as input of the maximum weight b-matching problem which resolution returns an optimal solution of $P \mid \text{LPSTIP}, p_j^{\min}=p^{\min}, d_j=d \mid -\sum w_j(p_j)$ when n_1 and t are fixed

Lemma I.7.2.1
Let $\mathcal{U}$ be the set of solutions such that
(1) jobs are scheduled in non-decreasing order of their deadlines on each machine and without idle time;
(2) only the last job scheduled on each machine has a processing time different from $p^{\min}$ (strictly greater);
(3) the number of jobs scheduled on each machine differs by at most 1.
$\mathcal{U}$ is dominant.

Proof. Figure I.10 illustrates the structure of the solutions of $\mathcal{U}$. Jobs with diagonal stripes are scheduled during $p^{\min}$.
Let S be an optimal solution. By Lemma I.3.1, we already proved that S can be modified to satisfy and (1).
(2) is also clear: if a job is scheduled during strictly less than $p^{\min}$, it can be removed without degrading the solution value, and if it is scheduled during strictly more than $p^{\min}$, its processing time can be reduced to $p^{\min}$, all the

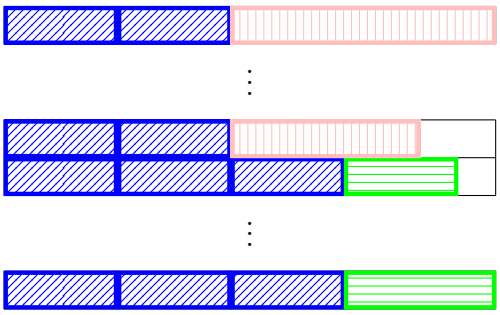


Figure I.10 – Illustration of the structure of the solutions of $\mathcal{U}$

following jobs on the machine are shifted toward the beginning and the last job processing time can be increased by the same quantity. Since $b_j = b$, the solution value is not degraded. The solution is still without idle time and the order does not change.

To prove (3), let s^i_{last} denote the start date of the last job scheduled on machine M_i. Now let M_{i_1} and M_{i_2} be two machines such that $s^{i_1}_{\text{last}} = \min_i s^i_{\text{last}}$ and $s^{i_2}_{\text{last}} = \max_i s^i_{\text{last}}$. If $s^{i_2}_{\text{last}} - s^{i_1}_{\text{last}} \leq p^{\min}$, then (3) is satisfied. Otherwise, let T_j be the last job scheduled on M_{i_1} and T_k be the job scheduled on M_{i_2} on time interval $\left[s^{i_1}_{\text{last}} + p^{\min}, s^{i_1}_{\text{last}} + 2p^{\min}\right[$. Let S' be the solution identical to S except that T_k is moved from M_{i_1} to M_{i_2} and these two machines are rescheduled as illustrated in Figure I.11. Thus, $w(M'_{i_1}) = w(M_{i_1}) - b(d_j - (s^{i_1}_{\text{last}} + p^{\min})) + w^{\min}_k + b(\max(d_j, d_k) - (s^{i_1}_{\text{last}} + 2p^{\min}))$ and $w(M'_{i_2}) = w(M_{i_2}) - w^{\min}_k + bp^{\min}$. Therefore, the value of the new solution is not degraded, the previous conditions are still satisfied, and the operation can be repeated till the condition is satisfied. ■

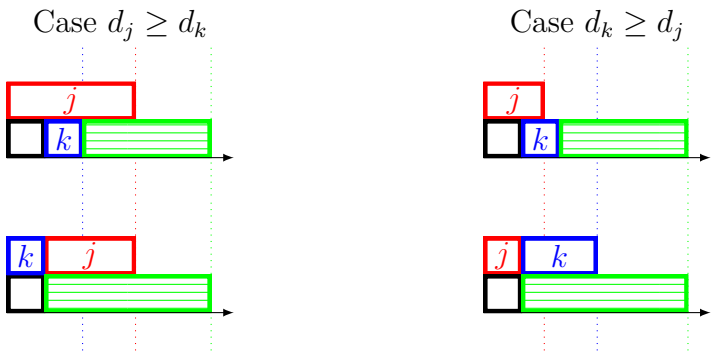


Figure I.11 – Operation which is repeated to make a solution satisfy (3)

Theorem I.7.2.1
There exists an algorithm that solves $P \mid \text{LPSTIP}, p_j^{\min}{=}p^{\min}, b_j{=}b \mid -\sum w_j(p_j)$ in polynomial time.

Proof. Using Lemma I.7.2.1, we only consider solutions with x machines with k jobs and $m-x$ machines with $k+1$ jobs. All jobs scheduled on a machine have a processing time of $p^{\min}$, except the last job of each machine which is scheduled till its deadline. If the number η of scheduled jobs is known, then k and x can be determined as the quotient and remainder from the euclidean division of η by m. When k and x are fixed, the problem can be model as a maximum weight b-matching problem: each job is associated to a node, as well as each possible position for a job and there is an arc between a job and a position if the job can be scheduled at this position (according to its deadline). The flow unit denotes wether or not a job is scheduled at a position.
A part of the graph including only one job is represented in Figure I.12. The size of the graph is polynomial compared to the size of the instance. Furthermore, n maximum weight b-matching problems have to be solved. Therefore, the problem can be solved in polynomial time. ■

I.7.3 $P \mid \textbf{LBPST}, p_j^{\max}{=}p^{\max}, d_j{=}d \mid -\sum w_j(p_j)$

With the same idea, we have:

Theorem I.7.3.1
There exists an algorithm that solves $P \mid \text{LBPST}, p_j^{\max}{=}p^{\max}, d_j{=}d \mid -\sum w_j(p_j)$ in polynomial time.

Proof. We consider $d = qp^{\max} + r$, with $q, r \in \mathbf{N}$, $0 \leq r < p^{\max}$.
Let $\mathcal{U}$ be the set of solutions such that:

- if $n \leq qm$, all n jobs are scheduled during $p^{\max}$;
- if $qm < n \leq (q+1)m$, all n jobs are scheduled; qm during $p^{\max}$ and $n - qm$ during r;
- if $n > (q+1)m$, $(q+1)m$ jobs are scheduled; qm during $p^{\max}$ and m during r.

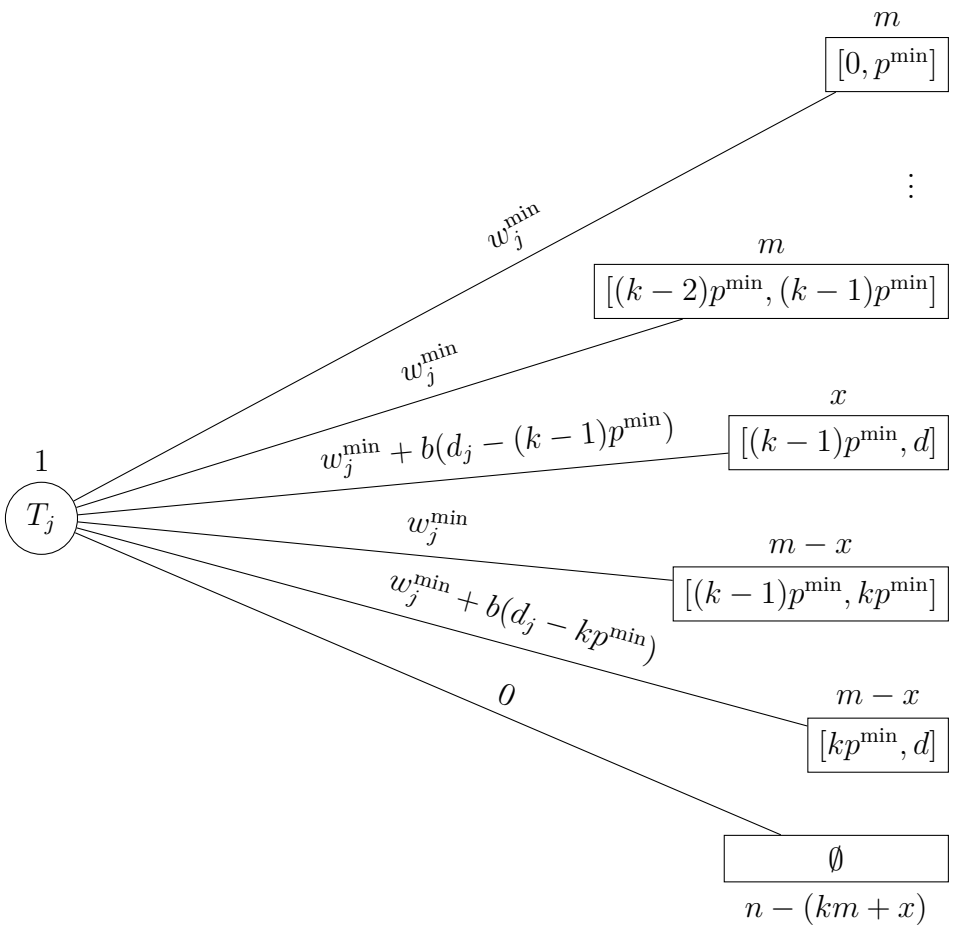


Figure I.12 – A part of the graph including only one job given as input of the maximum weight b-matching problem which resolution returns an optimal solution of $P \mid \text{LPSTIP}, p_j^{\min}{=}p^{\min}, b_j{=}b \mid -\sum w_j(p_j)$ when the number of scheduled jobs is fixed

Using similar arguments as before, one can prove that $\mathcal{U}$ is dominant.

Figure I.13 illustrates the structure of the solutions of $\mathcal{U}$ when $n > (q+1)m$. Jobs with diagonal stripes are scheduled during $p^{\max}$.

Now, if $n < qm$, all n jobs can be scheduled during $p^{\max}$ and this is an optimal solution. Otherwise we use a maximum weight b-matching model to solve the problem. We only present the case $n > (q+1)m$, but the model is similar for the case $qm < n \leq (q+1)m$. The model includes two groups of nodes: the nodes T_j, $j = 1, \ldots, n$, associated with jobs and the nodes U_1, U_2 and U_0 representing the state of a job in the solution (scheduled during $p^{\max}$, scheduled during r, or not scheduled). A part of the graph including only

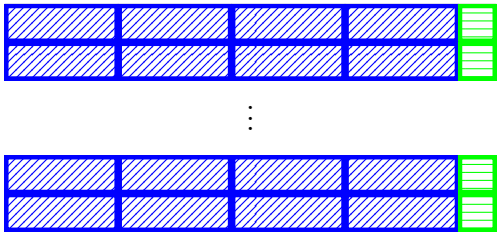


Figure I.13 – Illustration of the structure of the solutions of $\mathcal{U}$ when $n > (q+1)m$

one job is represented in Figure I.14. The size of the graph is polynomial compared to the size of the instance. Therefore, the problem can be solved in polynomial time. ■

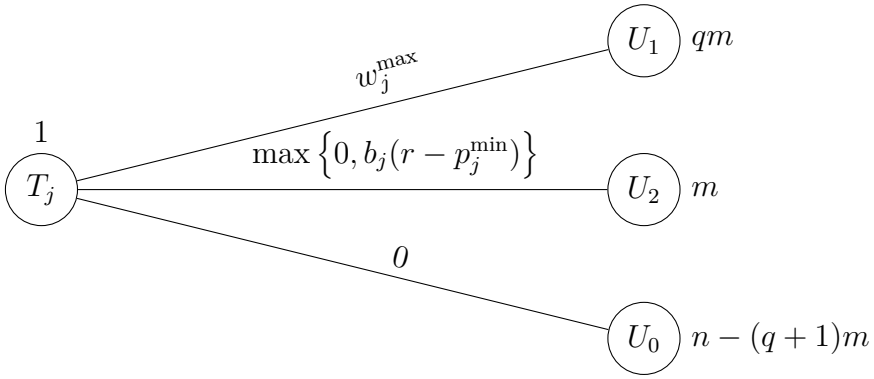


Figure I.14 – A part of the graph including only one job given as input of the maximum weight b-matching problem which resolution returns an optimal solution of $P \mid \text{LBPST}, p_j^{\max}{=}p^{\max}, d_j{=}d \mid -\sum w_j(p_j)$ when $n > (q+1)m$

I.8 Synthesis

In the previous sections, we studied different variants of PDPSP by resolution methods. In order to give the reader a model-based point of view, in this section, we summarize the results by profit functions.
Results are presented in Figures I.16 (LPST), I.18 (LBP) and I.20 (LPSTIP) and in Tables I.15 (LPST), I.17 (LBP), I.19 (LPSTIP), I.1 (LBPST) and I.2 (LBPSTIP).
In the Figures, each node corresponds to a problem. Rectangles indicate strongly NP-hard problems, chamfered rectangles indicate weakly NP-hard

problems, rounded rectangles indicate polynomial problems and tapes indicate problems which complexity is still open (see Figure I.21). Arrows indicate how the complexity changes when parameters are modified. If there is an arrow from variant $P1$ to variant $P2$, then $P2$ is "harder" than $P1$. For example, in Figure I.18 (LBP), the problem $P \mid w_j(p)=w(p)$ is still open. If the number of machines is bounded Pm, it becomes polynomial since $Pm \mid p_j^{\max}=p^{\max}$ is polynomial. If $p_j^{\max}$ take arbitrary values, then it is still open ($P \mid p_j^{\max}=p^{\max}$). Finally, if b_j (resp. $w_j^{\max}$) take arbitrary values, then it becomes strongly NP-complete since $P \mid b_j=b,\ d_j=d$ (resp. $P \mid w_j^{\max}=w^{\max},\ d_j=d$) is NP-complete.
In the Tables, "sNPc" indicate strongly NP-hard problems, "wNPc" indicate weakly NP-hard problems, "P" indicate polynomial problems and empty cells indicate problems which complexity is still open. Results in bold indicate maximally polynomial or minimally NP-complete cases for the corresponding profit function.
We do not consider very particular variants for which the instance size is $O(\log n)$ such as $P \mid$ LBPSTIP, $w_j(p)=w(p),\ d_j=d \mid -\sum w_j(p_j)$. These problems do not appear in the Figures and are noted $\times$ in the Tables. However, note that the instance size of $P \mid$ LBPSTIP, $w_j(p)=w(p) \mid -\sum w_j(p_j)$ is still $O(n)$ since deadlines are distinct.

For LPST, problems $P \mid$ LPST $\mid -\sum w_j(p_j)$ and $P \mid$ LPST, $p_j^{\min}=p^{\min} \mid -\sum w_j(p_j)$ are still open. All other cases are polynomial.
For LBP, all one machine cases are polynomial as well as some cases with parallel machines. The case with parallel machines and identical maximum processing-times is still open and interesting since it lies between $P \mid p_j=p \mid \sum w_jU_j$ and $P \mid p_j=p$, pmtn $\mid \sum w_jU_j$ that were proved to be respectively solvable in polynomial time and NP-complete (Brucker and Kravchenko 1999).
For LPSTIP, most cases with identical minimum processing times are polynomial, whereas having identical growth rates or minimum profits generally keeps the problem NP-complete. Three cases remain open.
For LBPST, many cases with identical maximum processing times are polynomial. Nine cases are still open.
Finally, for LBPSTIP, most cases are NP-complete and several problems remain open.

I.9 With release dates and preemption

In the previous sections, we looked at PDPS problems under various machine environments and with various profit functions, but without additional

$\mathcal{P}$	$P \mid$	$P \mid d_j{=}d$	$Pm \mid$	$Pm \mid d_j{=}d$	$1 \mid$	$1 \mid d_j{=}d$
		P	**P**	P	P	P
$b_j{=}b$	**P**	P	P	P	P	P
$p_j^{\min}{=}p^{\min}$		P	P	P	P	P
$w_j(p){=}w(p)$	P	×	P	×	P	×

Table I.15 – LPST

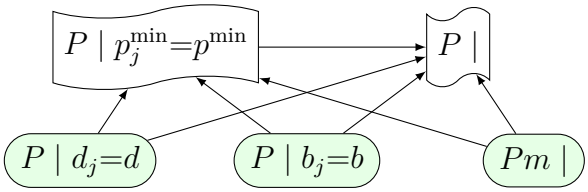


Figure I.16 – LPST

$\mathcal{P}$	$P \mid$	$P \mid d_j{=}d$	$Pm \mid$	$Pm \mid d_j{=}d$	$1 \mid$	$1 \mid d_j{=}d$
	sNPc	sNPc	wNPc	wNPc	**P**	P
$b_j{=}b$	sNPc	**sNPc**	wNPc	**wNPc**	P	P
$p_j^{\max}{=}p^{\max}$		**P**	**P**	P	P	P
$w_j^{\max}{=}w^{\max}$	sNPc	**sNPc**	wNPc	**wNPc**	P	P
$w_j(p){=}w(p)$		×	P	×	P	×

Table I.17 – LBP

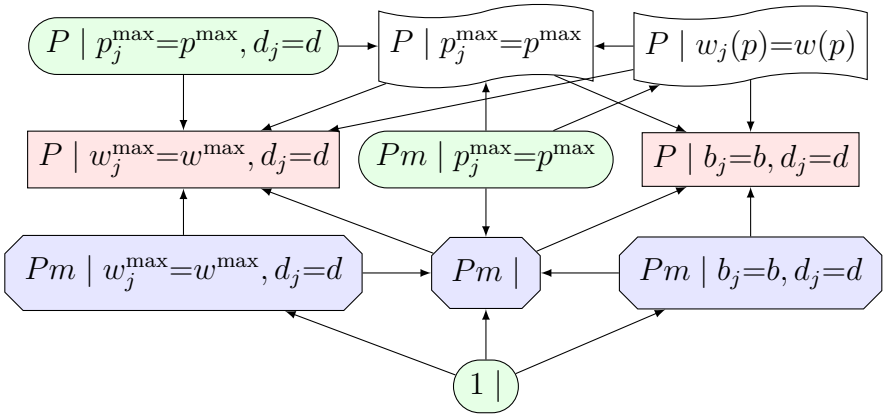


Figure I.18 – LBP

$\mathcal{P}$	$P \mid$	$P \mid d_j{=}d$	$Pm \mid$	$Pm \mid d_j{=}d$	$1 \mid$	$1 \mid d_j{=}d$
	sNPc	sNPc	wNPc	wNPc	wNPc	wNPc
$p_j^{\min}{=}p^{\min}$		**P**	**P**	P	P	P
$w_j^{\min}{=}w^{\min}$	sNPc	sNPc	wNPc	wNPc		**P**
$b_j{=}b$	sNPc	sNPc	wNPc	wNPc	wNPc	**wNPc**
$p_j^{\min}{=}p^{\min}, w_j^{\min}{=}w^{\min}$		P	P	P	P	P
$p_j^{\min}{=}p^{\min}, b_j{=}b$	**P**	P	P	P	P	P
$w_j^{\min}{=}w^{\min}, b_j{=}b$	sNPc	**sNPc**	wNPc	**wNPc**	**P**	P
$w_j(p){=}w(p)$	P	×	P	×	P	×

Table I.19 – LPSTIP

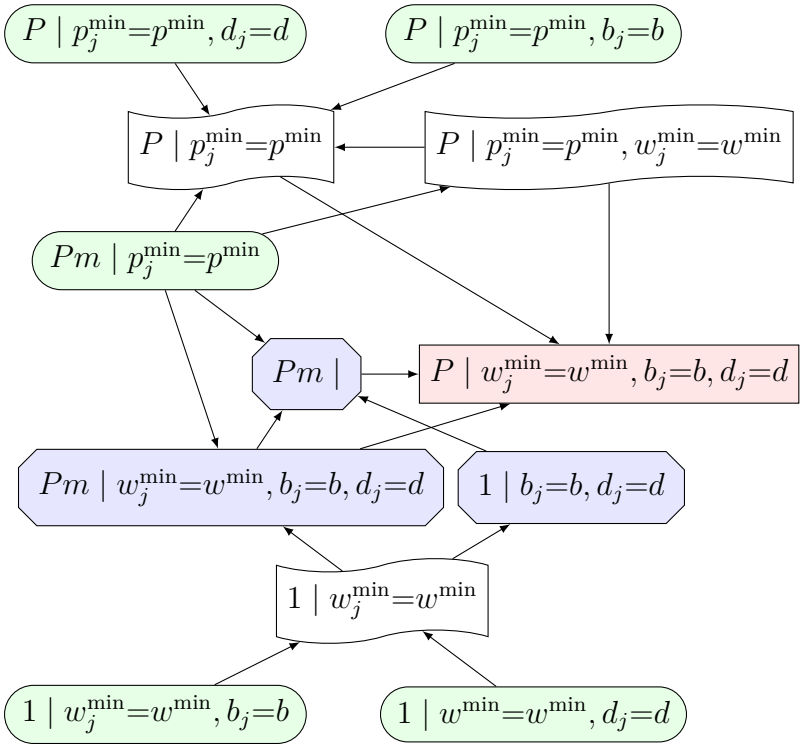


Figure I.20 – LPSTIP

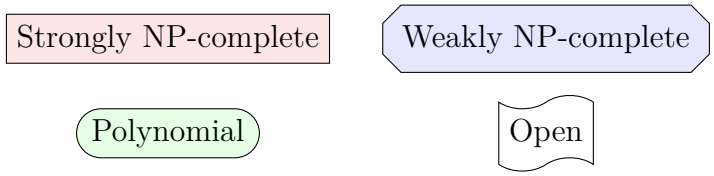


Figure I.21 – Figure caption

$\mathcal{P}$	$P \mid$	$P \mid d_j{=}d$	$Pm \mid$	$Pm \mid d_j{=}d$	$1 \mid$	$1 \mid d_j{=}d$
	sNPc	sNPc	wNPc	wNPc	wNPc	wNPc
$p_j^{\min}{=}p^{\min}$	sNPc	sNPc	wNPc	wNPc		
$p_j^{\max}{=}p^{\max}$		**P**	P	P	P	P
$b_j{=}b$	sNPc	sNPc	wNPc	wNPc	wNPc	wNPc
$w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc		P
$p_j^{\min}{=}p^{\min}, p_j^{\max}{=}p^{\max}$		P	P	P	P	P
$p_j^{\min}{=}p^{\min}, b_j{=}b$	sNPc	**sNPc**	wNPc	**wNPc**	**P**	P
$p_j^{\min}{=}p^{\min}, w_j^{\max}{=}w^{\max}$	sNPc	**sNPc**	wNPc	**wNPc**		P
$p_j^{\max}{=}p^{\max}, b_j{=}b$		P	P	P	P	P
$p_j^{\max}{=}p^{\max}, w_j^{\max}{=}w^{\max}$		P	P	P	P	P
$b_j{=}b, w_j^{\max}{=}w^{\max}$	sNPc	**sNPc**	wNPc	**wNPc**	P	P
$w_j(p){=}w(p)$		$\times$	P	$\times$	P	$\times$

Table I.1 – LBPST

$\mathcal{P}$	$P \mid$	$P \mid d_j{=}d$	$Pm \mid$	$Pm \mid d_j{=}d$	$1 \mid$	$1 \mid d_j{=}d$
$p_j^{\min}{=}p^{\min}, w_j^{\min}{=}w^{\min}$	sNPc	sNPc	wNPc	wNPc		
$p_j^{\min}{=}p^{\min}, b_j{=}b$	sNPc	sNPc	wNPc	wNPc		
$p_j^{\min}{=}p^{\min}, p_j^{\max}{=}p^{\max}$			P	P	P	P
$p_j^{\min}{=}p^{\min}, w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc		
$w_j^{\min}{=}w^{\min}, w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc		P
$w_j^{\min}{=}w^{\min}, b_j{=}b$	sNPc	sNPc	wNPc	wNPc	wNPc	wNPc
$w_j^{\min}{=}w^{\min}, p_j^{\max}{=}p^{\max}$	sNPc	sNPc	wNPc	wNPc		
$b_j{=}b, p_j^{\max}{=}p^{\max}$	sNPc	sNPc	wNPc	wNPc	wNPc	wNPc
$b_j{=}b, w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc	wNPc	**wPNc**
$w_j^{\max}{=}w^{\max}, p_j^{\max}{=}p^{\max}$	sNPc	sNPc	wNPc	wNPc	wNPc	**wNPc**
$p_j^{\min}{=}p^{\min}, w_j^{\min}{=}w^{\min}, b_j{=}b$	sNPc	sNPc	wNPc	wNPc		P
$p_j^{\min}{=}p^{\min}, w_j^{\min}{=}w^{\min}, p_j^{\max}{=}p^{\max}$			P	P	P	P
$p_j^{\min}{=}p^{\min}, w_j^{\min}{=}w^{\min}, w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc		P
$p_j^{\min}{=}p^{\min}, b_j{=}b, p_j^{\max}{=}p^{\max}$			P	P	P	P
$p_j^{\min}{=}p^{\min}, b_j{=}b, w_j^{\max}{=}w^{\max}$						P
$p_j^{\min}{=}p^{\min}, p_j^{\max}{=}p^{\max}, w_j^{\max}{=}w^{\max}$			P	P	P	P
$w_j^{\min}{=}w^{\min}, b_j{=}b, w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc	P	P
$w_j^{\min}{=}w^{\min}, b_j{=}b, p_j^{\max}{=}p^{\max}$	sNPc	sNPc	wNPc	wNPc		P
$w_j^{\min}{=}w^{\min}, p_j^{\max}{=}p^{\max}, w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc		P
$b_j{=}b, p_j^{\max}{=}p^{\max}, w_j^{\max}{=}w^{\max}$	sNPc	sNPc	wNPc	wNPc		P
$w_j(p){=}w(p)$		×	P	×	P	×

Table I.2 – LBPSTIP

constraints. In this section, we briefly glance at some cases where jobs have release dates or where preemption is allowed.

I.9.1 Preliminary results and notations

First, the following proposition rules out the interest of preemption for some 1-machine problems:

Proposition I.9.1.1
For $1 \mid \text{pmtn} \mid -\sum w_j(p_j)$, non preemptive solutions are dominant.

Note that this does not hold with parallel machines or with release dates.
If release dates are present but deadlines are identical, the problem is equivalent to the one without release dates and with deadlines $d'_j = d - r_j$.
When there are distinct release dates and distinct deadlines, as illustrated in Figure I.2 of Section I.1, we remind that solutions containing all jobs are not dominant (without preemption). In this case, we define the ordered set $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{2n}\}$ as the union of all release dates $\{r_j\}_{j=1\ldots n}$ and all deadlines $\{d_j\}_{j=1\ldots n}$, sorted in non-decreasing order; we then derive the ordered set $\mathcal{J}$ of intervals as:

$$\mathcal{J} = \{[\alpha_t, \alpha_{t+1}]\}_{t=1\ldots 2n-1}$$

I.9.2 Linear profits (LP)

For the linear model, all cases were polynomial. If release dates are present and preemption is allowed, then scheduling the best job at each instant leads to an optimal solution (this even allows for multiple time-windows). This can be achieved by scheduling jobs in non-increasing order of their growth rates on their whole remaining time-windows as described in Algorithm 12, and it is a special case of Theorem I.9.3.2.

Algorithm 12 $1 \mid \text{LP}, r_j, \text{pmtn} \mid -\sum w_j(p_j)$

```
Sort jobs in non-increasing order of b_j.
for j from 1 to n do
    for J ∈ 𝒥 do
        if J ⊆ [r_j, d_j[ and no job is yet scheduled on J then
            Schedule T_j on J
```

However, without preemption, the problem is still open, even if some special cases are easy.

First, note that the set of solutions for which the start date and the end date of each task belong to A is dominant. More precisely, the set of solutions such that for each $j = 1, \ldots, n$, T_j starts either at r_j or at the deadline d_k of the previous job T_k without idle time (if it exists), and ends either at d_j or at the release date r_k of the following job T_k without idle time (if it exists), is dominant.
With identical growth rates, the problem becomes an uptime maximization and it is enough to schedule jobs as soon as they are available as long as possible:

Theorem I.9.2.1
Scheduling jobs in non-decreasing order of their release dates, and until their deadlines (when possible), leads to an optimal solution of $1 \mid \text{LP}, r_j, b_j{=}b \mid -\sum w_j(p_j)$ in polynomial time.

If no job time-window is strictly included in the time-windows of another job with a strictly greater growth rate, then Algorithm 12 used when preemption is allowed will schedule each job on consecutive intervals, and thus lead to a feasible and therefore optimal solution.

Theorem I.9.2.2
Let us call T_j a splitting job iff there exists T_k such that $r_j > r_k$, $d_j < d_k$ and $b_j > b_k$. If there is no splitting jobs, then Algorithm 12 returns a feasible optimal solution for $1 \mid \text{LP}, r_j \mid -\sum w_j(p_j)$.

For the general case, we know that a job which is the best on its time-window will either be scheduled on its whole time-window or not be scheduled at all, *i.e.* if for all $k \neq j$, $[r_j, d_j[\cap [r_k, d_k[\neq \emptyset \implies b_j \geq b_k$, then there exists an optimal solution for which T_j is either scheduled on its whole time-window or not scheduled at all. This dominance property gives a possibility for a branch-and-bound algorithm scheduling jobs in non-decreasing order of their growth rates. Note that a job may then have multiple time-windows. The algorithm runs in exponential time, but if the number of splitting jobs k is bounded, then it runs in polynomial time $O((nk)^k)$.

I.9.3 Linear bounded profits (LBP)

Adding release dates makes the problem NP-complete:

Theorem I.9.3.1
$1 \mid r_j \mid -\sum w_j(p_j)$ is strongly NP-complete with the following profit functions:

(1) LBP, b_j=b

(2) LBP, $w_j^{\max}$=$w^{\max}$

Proof. First, note that the problem clearly belongs to NP. Then, we shall prove that 3-PARTITION, which is known to be strongly NP-complete (Garey and Johnson 1979), reduces to the decision version.
Let I be an instance of 3-PARTITION: a finite set A of $3m$ elements, a bound $B \in \mathbf{Z}^+$, and a size $s(a) \in \mathbf{Z}^+$ for each $a \in A$, such that each $s(a)$ satisfies $B/4 < s(a) < B/2$ and such that $\sum_{a\in A} s(a) = mB$; the question is: can A be partitioned into m disjoint sets $S_1, \ldots, S_m$, such that for $1 \leq i \leq m$, $\sum_{a\in S_i} s(a) = B$?

(1): Let I' be an instance of $1 \mid \text{LBP}, b_j{=}b, r_j \mid -\sum w_j(p_j)$ with $3m$ "real" jobs T_a^1, $a \in A$, such that $r_a^1 = 0$, $d_a^1 = d$ (with $d = mB + (m-1)$), $b_a^1 = b$, $p^{\max 1}_a = s(a)$, and $m-1$ "dummy" jobs T_j^0, $1 \leq j \leq m-1$, such that $r_j^0 = j(B+1) - 1$, $d_j^0 = j(B+1)$, $p^{\max 0}_j = 1$, $b_j^0 = b$; the question is: is there a schedule S such that $w(S) = bd$?
I responds "yes" clearly implies a "yes" answer to I'. Conversely, I' responds "yes" means that there is a schedule with no idle time in which each job is scheduled and executed till its maximum processing-time. In particular, each job T_j^0, $1 \leq j \leq m-1$ is scheduled on $[j(B+1)-1, j(B+1)[$, which means that they partition the other jobs into sets with processing-time B.

(2): for the case with identical maximum profit $w_j^{\max}$=$w^{\max}$, we consider the same reduction, but the maximum profit of the jobs of I' is $w^{\max}$ and b_j are set accordingly; the question is: Is there a schedule S such that $w(S) = (3m + (m-1))w$?

In all cases, the transformations are not only polynomial but even strongly polynomial; hence the strong NP-completeness of the concerned cases. ■

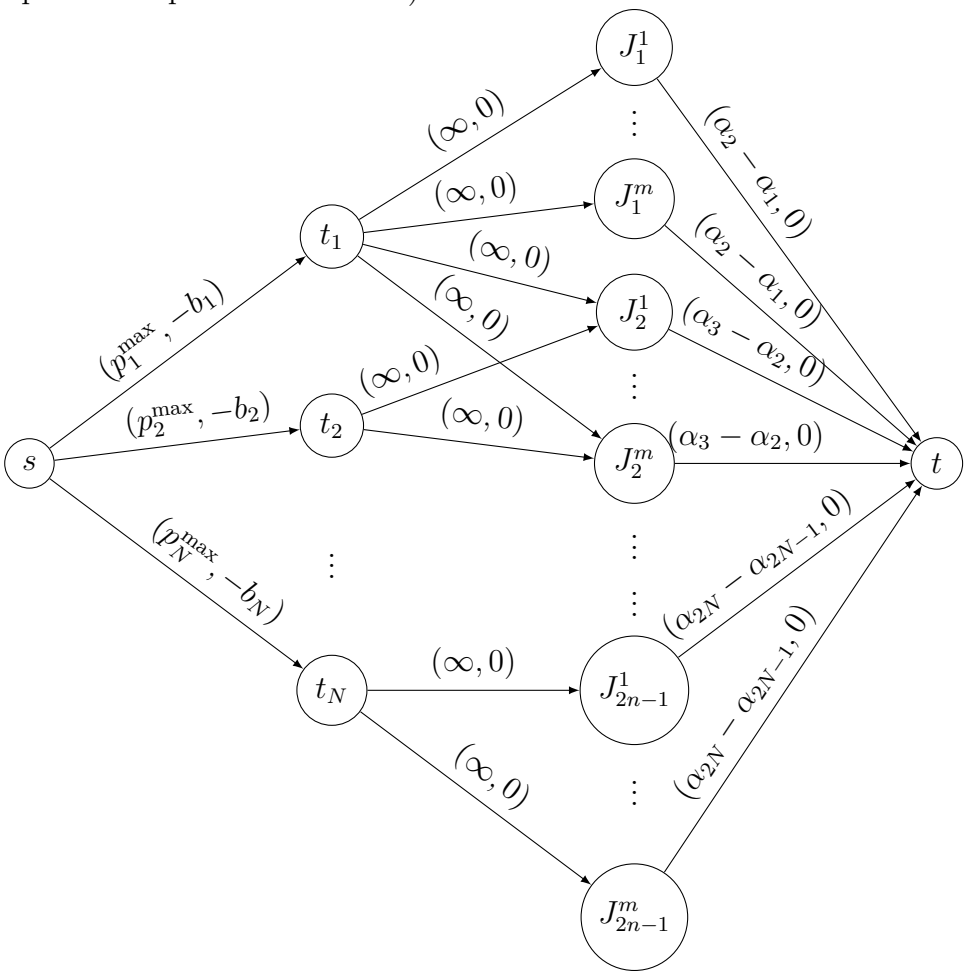
However, if preemption is allowed, the problem becomes polynomially solvable, even with parallel machines. Indeed, the problem can be seen as the scheduling problem with late jobs $P \mid \text{pmtn}, r_j \mid \sum w_j Y_j$. This problem can be solved with a minimum cost flow model in polynomial time (Leung et al. 2004):

Theorem I.9.3.2
Algorithm 13 returns an optimal solution for $P \mid \text{LBP}, r_j, \text{pmtn} \mid -\sum w_j(p_j)$ in polynomial time.

Proof. Algorithm 13 illustrates the flow model with the notations of our problem. Let us recall that $\mathcal{J}$ is the set of ordered time-intervals defined by release dates and deadlines.
There is a node t_j, $j = 1, \ldots, n$, for each job T_j and a node J_t^i, $t = 1, \ldots, 2n-1$, $i = 1, \ldots, m$, for each element of $\mathcal{J} \times \{1, \ldots, m\}$. There is an arc from the source s to each t_j, $j = 1, \ldots, n$ with capacity $p_j^{\max}$ and cost $-b_j$, an arc from each t_j, $j = 1, \ldots, n$, to each J_t^i if $[\alpha_t, \alpha_{t+1}[\subseteq [r_j, d_j[$ with an infinite capacity and no cost, and an arc between each J_t^i, $t = 1, \ldots, 2n-1$, $i = 1, \ldots, m$, and the sink t with capacity $\alpha_{t+1} - \alpha_t$ and no cost. ■

Algorithm 13 1 | LBP, r_j, pmtn | $-\sum w_j(p_j)$

1: Solve the following minimum-cost flow problem (the pairs on the edges correspond to capacities and costs):



2: Use a wrap around to convert the flow solution into a feasible schedule.

Note that with only few changes, this flow model can be adapted to jobs with disjoint time-windows and to any concave piecewise profit function.

I.9.4 Linear profits with setup times (LPST)

The general cases with release dates and with or without preemption are still open. However, a dynamic programming algorithm solves the problem if growth rates are identical. This algorithm is similar to Algorithm 1 except that only the release dates are considered for start dates:

Theorem I.9.4.1
Dynamic programming Algorithm 14 returns the optimal value of $Pm \mid \text{LPST}, b_j{=}b, r_j \mid -\sum w_j(p_j)$ in polynomial time $O(n \log n + m^2 n^m)$.

Algorithm 14 $P \mid \text{LPST}, b_j{=}b, r_j \mid -\sum w_j(p_j)$

1: Sort tasks in non-decreasing order of their deadlines
2: Return $f^*(n, (d_n, d_{n-1}, \ldots, d_{n-m+1}))$ with

$$f^*(j, C) = \begin{cases} 0 & \text{if } j = 0 \\ \max \begin{cases} f^*(j-1, C) \\ \max\limits_{\substack{1 \le i \le m \\ C_i \ge r_j\}}} f^*(j-1, C - (C_i - r_j)e^i) + w_j(\min\{C_i, d_j\} - r_j) \end{cases} & \text{otherwise} \end{cases}$$

where C is a m-dimensional vector containing the required completion time of the last task scheduled on each machine and e^i is the vector such that $e^i_i = 1$ and $e^i_k = 0$ for all $k \neq i$.

Proof. This proof being very similar to the previous proofs of this chapter, we will be more concise. Even though Lemma I.3.1 does not apply with release dates, scheduling jobs in non-decreasing order of their deadlines remains dominant: indeed with LPST with identical growth rates, it is dominant to end jobs by order of their deadlines and therefore to schedule them in this order. As a consequence, Algorithm 1 would return the optimal value of the problem. To have an efficient algorithm, we consider the set $\mathcal{U}$ of solutions such that all scheduled jobs are executed in non-increasing order of their deadlines and start at their release date; this set is dominant. Thus, for each job, only one start date is considered in Algorithm 14 and the size of C is n^m. The overall complexity is thus $O(n \log n + m^2 n^m)$. ■

Note that even if preemption is allowed, non preemptive schedules are dominant. Therefore, the optimal value is the same as in the non preemptive case.

I.10 Conclusions

In this chapter, we studied a category of scheduling problems which are of theoretical and practical interest: processing time dependent profit maximization scheduling problems. We discussed some unusual properties and we proposed several piecewise-linear models as references. Then we underlined the complexity hierarchy that exists among those models. We proved NP-completeness of most general cases. Using various technics, we proposed dedicated polynomial algorithms for many particular cases; drawing a first map of these territories.
In the process, the complexity of many cases, including that of an original astrophysics application, have been derived. However, we merely laid some foundations, and many cases remain open or incomplete, and new models of profit functions or other constraints are still to be studied.

References

Bartal, Leonardi, Marchetti-Spaccamela, Sgall, and Stougie (2000). "Multiprocessor Scheduling with Rejection". In: *SIAM Journal on Discrete Mathematics* 13.1, pp. 64–78. DOI: 10.1137/S0895480196300522.

Brucker and Kravchenko (1999). *Preemption Can Make Parallel Machine Scheduling Problems Hard*.

Cao, Wang, Zhang, and Liu (2006). "On Several Scheduling Problems with Rejection or Discretely Compressible Processing Times". en. In: *Theory and Applications of Models of Computation*. Ed. by Jin-Yi Cai, S. Barry Cooper, and Angsheng Li. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 90–98.

Catusse, Cambazard, Brauner, Lemaire, Penz, Lagrange, and Rubini (2016). "A Branch-And-Price Algorithm for Scheduling Observations on a Telescope". In: *Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI Press, pp. 3060–3066.

Cordeau and Laporte (2005). "Maximizing the value of an Earth observation satellite orbit". en. In: *Journal of the Operational Research Society* 56.8, pp. 962–968. DOI: 10.1057/palgrave.jors.2601926.

Dantzig (1957). "Discrete-Variable Extremum Problems". In: *Operations Research* 5.2, pp. 266–288. DOI: 10.1287/opre.5.2.266.

Engels, Karger, Kolliopoulos, Sengupta, Uma, and Wein (2003). "Techniques for scheduling with rejection". In: *Journal of Algorithms*. 1998 European Symposium on Algorithms 49.1, pp. 175–191. DOI: 10.1016/S0196-6774(03)00078-6.

Eun, Sung, and Kim (2017). "Maximizing total job value on a single machine with job selection". en. In: *Journal of the Operational Research Society* 68.9, pp. 998–1005. DOI: 10.1057/s41274-017-0238-z.

Garey and Johnson (1978). "" Strong " NP-Completeness Results: Motivation, Examples, and Implications". In: *J. ACM* 25.3, pp. 499–508. DOI: 10.1145/322077.322090.

Garey and Johnson (1979). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Habet, Vasquez, and Vimont (2010). "Bounding the optimum for the problem of scheduling the photographs of an Agile Earth Observing Satellite". en. In: *Computational Optimization and Applications* 47.2, pp. 307–333. DOI: 10.1007/s10589-008-9220-7.

Janiak and Kovalyov (1996). "Single machine scheduling subject to deadlines and resource dependent processing times". In: *European Journal of Operational Research* 94.2, pp. 284–291. DOI: 10.1016/0377-2217(96)00129-4.

Kellerer, Pferschy, and Pisinger (2004). "Advanced Algorithmic Concepts". en. In: *Knapsack Problems*. Springer, Berlin, Heidelberg, pp. 43–72. DOI: 10.1007/978-3-540-24777-7_3.

Lagrange, Rubini, Brauner, Cambazard, Catusse, Lemaire, and Baude (2016). "SPOT: an optimization software for dynamic observation programming". In: *SPIE 9910, Observatory Operations: Strategies, Processes, and Systems VI, 991033 (July 18, 2016)*. Edinburgh, United Kingdom. DOI: 10.1117/12.2241197..

Lawler (1976). "Sequencing to Minimize the Weighted Number of Tardy Jobs". In: *Revue Française d'Automatique, Informatique, Recherche Opérationnelle. Série Bleue* 10.

Leung, Kelly, and Anderson (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, Inc.

Lin and Ying (2015). "Order acceptance and scheduling to maximize total net revenue in permutation flowshops with weighted tardiness". In: *Applied Soft Computing* 30, pp. 462–474. DOI: 10.1016/j.asoc.2015.01.069.

Pinedo (2015). *Scheduling*. Springer.

Potts and Van Wassenhove (1988). "Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs". In: *Management Science* 34.7, pp. 843–858.

Schrijver (2002). *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Springer Science & Business Media.

Shabtay, Gaspar, and Kaspi (2013). "A survey on offline scheduling with rejection". en. In: *Journal of Scheduling* 16.1, pp. 3–28. DOI: 10.1007/s10951-012-0303-z.

Shabtay, Gaspar, and Yedidsion (2012). "A bicriteria approach to scheduling a single machine with job rejection and positional penalties". en. In: *Journal of Combinatorial Optimization* 23.4, pp. 395–424. DOI: 10.1007/s10878-010-9350-6.

Shabtay and Steiner (2007). "A survey of scheduling with controllable processing times". In: *Discrete Applied Mathematics* 155.13, pp. 1643–1666. DOI: 10.1016/j.dam.2007.02.003.

Shioura, Shakhlevich, and Strusevich (2018). "Preemptive models of scheduling with controllable processing times and of scheduling with imprecise computation: A review of solution approaches". In: *European Journal of Operational Research* 266.3, pp. 795–818. DOI: 10.1016/j.ejor.2017.08.034.

Slotnick (2011). "Order acceptance and scheduling: A taxonomy and review". In: *European Journal of Operational Research* 212.1, pp. 1–11. DOI: 10.1016/j.ejor.2010.09.042.

Sterna (2011). "A survey of scheduling problems with late work criteria". In: *Omega* 39.2, pp. 120–129. DOI: 10.1016/j.omega.2010.06.006.

Tangpattanakul, Jozefowiez, and Lopez (2015). "A multi-objective local search heuristic for scheduling Earth observations taken by an agile satellite". In: *European Journal of Operational Research* 245.2, pp. 542–554. DOI: 10.1016/j.ejor.2015.03.011.

Verfaillie and Lemaître (2001). "Selecting and Scheduling Observations for Agile Satellites: Some Lessons from the Constraint Reasoning Community Point of View". en. In: *Principles and Practice of Constraint Programming — CP 2001*. Ed. by Toby Walsh. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 670–684.

Wang, Reinelt, Gao, and Tan (2011). "A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation". In: *Computers & Industrial Engineering*. Combinatorial Optimizatiion in Industrial Engineering 61.2, pp. 322–335. DOI: 10.1016/j.cie.2011.02.015.

Yang and Geunes (2007). "A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times". In:

Computers & Industrial Engineering 53.3, pp. 420–432. DOI: 10.1016/j.cie.2007.02.005.

Zhang, Lu, and Yuan (2009). "Single machine scheduling with release dates and rejection". In: *European Journal of Operational Research* 198.3, pp. 975–978. DOI: 10.1016/j.ejor.2008.10.006.

– (2010). "Single-machine scheduling under the job rejection constraint". In: *Theoretical Computer Science* 411.16, pp. 1877–1882. DOI: 10.1016/j.tcs.2010.02.006.

Chapter II

A fast, efficient, simple and versatile Large Neighborhood Search algorithm to schedule star observations on the Very Large Telescope

Large telescopes are few and observing time is a precious resource subject to a strong pressure from a competitive community. Besides, a growing number of astronomical projects require surveying a large number of targets during discontinuous runs spread over few months or years. An important example in today astronomy is the imaging surveys to discover and study extra-solar planets, which require to observe hundreds of targets to get a chance of imaging a few planets. There are many targets to be selected as actual observations and the total demand of observing time far exceeds the supply. Furthermore, the optimization must occur on different scales: prior to each year or semester, the requested targets list must fit the projected calendar, leading to adjust both; then the schedule is optimized again prior to each run (roughly, every week) and re-optimized prior to each night, to take into account priority changes and to reschedule failed observations. Even during one night, unexpected conditions such as bad weather might indeed require quasi real time adjustments. Therefore, efficient and both short-term and long-term (*i.e.* anytime) algorithms need to be designed.[1]

II.1 Problem description

The first part of this work has been conducted by Catusse et al. 2016. They studied a simplified model, its complexity, and developed algorithms to solve it. We first introduce their model: let $\mathcal{M}$ be a set of m nights and $\mathcal{T}$ be a set of n targets. Each target $T_j \in \mathcal{T}$ has a profit w_j and is visible during a

[1] Introductory paragraph from Catusse et al. 2016 written by my supervisors.

set of nights $\mathcal{M}_j$, and for each night $i \in \mathcal{M}_j$, T_j has a time-window $[r_j^i, d_j^i[$ and a duration p_j^i such that $2p_j \geq d_j^i - r_j^i$ (for almost all targets, there is a compulsory observation instant, the meridian). The objective is to maximize the total profit of the observed targets, such that each scheduled target T_j must be observed within one of its time-windows during the corresponding observation time, and only one target can be observed at a time.
Usually, a target is not visible in all nights, but rather in few sets of consecutive nights, and the time-window is slightly shifted at each night. Real instances include up to 1000 targets to schedule in about 100 nights. Eventually, between four and eight observations are scheduled in a night. The profit of an observation lies between 10 and 40.
In the remaining of the chapter, we call this problem the Pure Star Observation Scheduling Problem (Pure SOSP).
To solve this problem, Catusse et al. 2016 propose a branch-and-price algorithm combined with constraint programming, and a local search algorithm. They evaluated the performances of these algorithms on a dataset including one real instance and other generated instances of different sizes built to reproduce the properties of the real one. The branch-and-price algorithm is able to solve to optimality almost all tested instances within two hours whereas the local search algorithm provides solutions of good quality within a few minutes. The limited time-window size ($2p_j^i \geq d_j^i - r_j^i$) is a key feature of the problem resolution and is compulsory for the kind of images taken. It is equivalent for observations to having a "mandatory" instant, and therefore, it imposes the order for the observations scheduled within a night.
Even if the model described above is only a simplified version of the actual practical problem, the work carried by Catusse et al. 2016 shows that the problem can be handled and therefore it is worth refining the model to get more realistic solutions. It also gives bounds that can be used to evaluate the performances of algorithms developed for richer models. From the users point of view, it also helped the astrophysicists to understand how operations research can be of great help and how to model more efficiently the problem.

In this chapter, we therefore consider a more accurate version of the model. The refinements are the following:

- The duration of an observation actually depends on its start date. Usually, the duration is the smallest around the middle of the time window and slightly increases if it is done before or after.
- Furthermore, observation durations are flexible. If an observation is slightly shorter, then the picture will be of lesser quality, but shortening an observation may be worth if it makes room for other ones. Note

that due to telescope calibration, the observation always needs at least some time before a first valuable picture can be obtained; and beyond a certain duration, the picture quality stops increasing. This model (without the start date dependence) has been studied in Chapter I (see Figure I.3).

- Some observations called calibration observations are mandatory. They typically represent between 3% and 10% of the total number of observations scheduled.

- The mandatory instant property remains true for most observations, but some of them do not satisfy it. This may be for example due to the flexibility of observation durations, but also, calibration observations usually have very large time windows.

- In a few cases, a target might have several time windows during the same night. In the case of calibration observations, these time windows can even overlap.

- Besides scientific interests, targets also have emergencies. Some targets might be important to observe earlier in the schedule.

Currently, the astrophysicists schedule their observations using a software named SPOT (Lagrange et al. 2016). This software takes as input a list of stars with their astronomical properties, computes the astrophysical parameters (observation times, visiblity windows...), and does the optimization. The optimization part is achieved by an algorithm based on Simulated Annealing. It returns rather good solutions, but its convergence is rather slow. The goal of the present work is to propose to the astrophysicists a fast and efficient algorithm than can handle real problems, and to implement it in the software SPOT. As a consequence, the following practical constraints are also to be taken into account:

- As explained in the first paragraph, the algorithm should be anytime, providing good solutions in a few seconds or minutes, but still improving them if run several hours. Typically, astrophysicists want to have a good idea of the solution quality in order to, for example, adjust the profits. But then they will let the algorithm run during the whole day to improve the solutions. Quick solutions of good quality are also important to handle unexpected events during the night (clouds, rain, earthquakes...) that require modifiying the schedule. Furthermore,

some artefacts from the software (observation time computing, visualization, *etc.*), significatively slows the algorithm. Therefore, quickly finding good solutions is even more crucial.

- Despite this, the optimal solution is not the absolute goal, and good industrial solutions may be sufficient. Profits are actually not perfectly accurate and astrophysicists will update them if they want to. However, the problem changes quickly, either because the constraints change, or because the astrophysicists may propose better models (for example the inclusion of emergencies). Therefore, it should be easy to update the algorithm to handle those changes.

- Eventually, the proposed algorithm should be simple: a developer with only little knowledge in operations research should be able to understand it, implement it and maintain it. This also implies avoiding the use of external solvers.

The branch-and-price proposed by Catusse et al. 2016, although highly effective, does not meet the above requirements. First, it is not anytime: the column generation procedure takes time and on some instances, several dozens of minutes are necessary to output the first solution. Second, it is not easy to adapt if the new model breaks the separable structure. Finally, it requires too many knowledge in discrete optimization and the use of a linear solver and a constraint programming solver.

Most papers published on related problems (see Section II.2) focus on a very specific variant of a problem and try to develop the best possible algorithms. This approach does not fit with our need of versatility and simplicity. Furthermore, it makes it difficult to really evaluate the performance of the algorithms. Therefore, we choose a different approach: our goal here is to design an efficient algorithm that can be applied to any variant of SOSP, such that little adaptation is required for each variant. We implemented and evaluated our algorithm on five variants of the problem:

- The Pure Star Observation Scheduling Problem (Pure SOSP)
- The Pure SOSP with discrete flexible observation times (Pure df-SOSP)
- The Pure SOSP with continuous flexible observation times (Pure cf-SOSP)
- IPAG SOSP
- IPAG SOSP with discrete flexible observation times (IPAG df-SOSP)

Pure SOSP has already been presented at the begninning of the section. Thanks to the bounds computed by Catusse et al. 2016, we may evaluate the performances of our algorithms and compare them with their Local Search.
Among all additional constraints cited above, only the flexible duration constraint is not currently implement in SPOT and is a new request from the astrophysicists. We wanted to evaluate the effect of this property on the solutions. Therefore, we considered two variants of Pure SOSP where the processing time can be reduced by a factor x. If $p_j(S)$ is the duration of observation T_j during night i in solution S, then, in the discrete variant (Pure df-SOSP), $p_j(S) \in \left\{xp_j^i, p_j^i\right\}$, and in the continuous version (Pure cf-SOSP), $p_j(S) \in \left[xp_j^i, p_j^i\right]$. Then the profit returned by the observation is $w_j(S) = w_j p_j(S)/p_j^i$. Note that all solutions of Pure SOSP are feasible solutions of Pure df-SOSP, and all solutions of Pure df-SOSP are feasible solutions of Pure cf-SOSP. Therefore, we expect to get solutions with greater values in those cases.
IPAG SOSP is the current problem solved by the software SPOT (IPAG, "Institut de Planétologie et d'Astrophysique de Grenoble", is the name of the astrophysicists' research unit). It includes all constraints except the flexible durations. By evaluating the performances of our algorithms on this problem, we can compare the results with the algorithm currently implemented and show the astrophysicists if we manage to get better solutions. Since a target may have several time-windows in the same nights, we note K_j the number of time-windows of target T_j, and for each time-window TW_j^k, $k = 1, \ldots, K_j$, we note r_j^k its release date, d_j^k its deadline, m_{jk} its night, w_{jk} its profit and $p_j^k(s)$ the require observation time when starting it at time s.
Finally, IPAG df-SOSP is the problem including all constraints, *i.e.* all constraints implemented in SPOT and the discrete flexible observation times. Similarly as above, if $p_j(S)$ is the duration of observation T_j in time-window k starting at time s in solution S, then $p_j(S) \in \left\{xp_j^k(s), p_j^k(s)\right\}$. Then the profit returned by the observation is $w_j(S) = p_j(S)/p_j^k(s) * w_j^k$.
We do not look at the continuous version of IPAG df-SOSP. Indeed, it is a bit more complicated to handle and in Section II.4, we observe that our algorithm returns equivalent solution values for both cf-SOSP and df-SOSP.

II.2 Literature review

Besides the work published by Catusse et al. 2016, star observation scheduling problems have not receive a lot of attention. Johnston 1990 presents Spike, a software for scheduling observations on Hubble Space Telescope based on

Constraint Programming. The constraints for a space telescope are not the same as the ones for an Earth telescope. Still, Johnston 1993 introduced the problem of scheduling star observations on the Very Large Telescope and states that Spike may be used to solve the problem. Unfortunately, no computational experiments are presented.

On the other hand, many papers have been published on Agile Earth Observation Satellites (AEOS) scheduling problems, looking in the opposite direction from the sky towards the Earth (whereas we look from the Earth to the sky). Still, the problems share many similarities. The idea behind AEOS scheduling problem is to schedule projects on a satellite orbiting around the Earth. Each possible project has a time-window and returns a profit. The objective is to maximize the total profit of the schedule. The problem was introduced by Hall and Magazine 1994 in a single machine version. Wolfe and Sorensen 2000 improved the model, considering that an observation is better when it is centered in the time window. This property is similar to the start date dependent observation duration of our problem. Then Verfaillie and Lemaître 2001 considered a model with observation time dependent profit. This property is also one that we added in our model.

The first model considering several machines was introduced by Dilkina and Havens 2005, by considering a fleet of satellites. Interestingly, having multiple satellites following each other implies that the time-windows will be shifted on each machine. A similar phenomenon happens with star observations: the time-windows of observations of a night are shifted in the next night. Bianchessi et al. 2007, Wang et al. 2011, Xu et al. 2016, He et al. 2018 and Peng et al. 2019 also considered multi-machine versions of the problem, either by considering multiple satellites, multiple orbits, or both.

Some papers also consider additional constraints that are not related to our problem such as download, energy consumption, transition times, uncertainties (Verfaillie and Lemaître 2001; Lemaître et al. 2002; Dilkina and Havens 2005; Li et al. 2007 *etc.*). Note that, as stated in the previous section, there are uncertainties in our problem, such as weather or earthquakes. Those uncertainties are managed by relaunching the algorithm, since the robustness of the schedule does not matter for the astrophysicists. This is one of the reason why it is crucial to be able to get good solution very rapidly.

Unfortunately, algorithms developed for AEOS scheduling problems cannot be used for SOSP. Indeed, in multi-machines AEOS scheduling problems, the number of machines is small, *i.e.* lesser than five. In our problem, the number of machines, *i.e.* the number of nights is at least several dozens and can overcome a hundred. Furthermore, the telescope only observes at night, whereas satellites observe continuously and star observations are longer than earth observations. Therefore, even if the two families of problems are similar,

AEOS scheduling problems include few machines but many jobs per machine, whereas star observation scheduling problems include many machines but few jobs per machine. It follows that algorithms developed for AEOS scheduling problems are not adequate for star observation scheduling problems.

II.3 Large neighborhood search

We propose a Large Neighorhood Search (LNS) to solve the problem. LNS is a well-known meta-heuristic that has been widely used in Operations Research (see Pisinger and Ropke 2010). In LNS, an initial solution is iteratively improved by destroying and repairing it. Here, we implemented it by emptying and rescheduling several nights together. The repair is performed with a heuristic greedy algorithm.
In the algorithm, a solution S consists of m single night schedules S_i, $i = 1, \ldots, m$. For a night $i \in \mathcal{M}$, S_i is defined by a list of tuples (j, k, τ, s, p), where T_j is the corresponding target, TW_j^k the corresponding time-window, s is the observation start date and p the observation duration. Tuples are sorted and the corresponding observations are scheduled in non-decreasing order of τ.

II.3.1 Single Night Timing Problem

A major interest of the proposed architecture is that in order to handle the different variants of SOSP considered, one just needs to implement an algorithm (exact or heuristic) for a sub-problem called the Single Night Timing Problem (SNTP). SNTP takes as input a list of observations belonging to the same night and computes whether it is possible to schedule all of them in the given order, and if yes, returns the corresponding profit and schedule. Algorithm 15 solves SNTP for Pure SOSP. It is rather straightforward. Since the order is imposed, one just needs to schedule the observations as early as possible. Solving the feasibility problem is actually equivalent for all variants: one just needs to schedule the observations as early as possible at their smallest possible duration. Therefore, in the remaining of the section, we only describe the algorithms that compute the schedule and its profit.

SNTP for Pure cf-SOSP is similar to the problem presented by Janiak and Kovalyov 1996. They studied the case without release dates and proposed a $O(n \log n)$ exact algorithm. Algorithm 16 is an adaptation of this algorithm in order to take into account the release dates. It is similar to Algorithm 10 presented in Chapter I page 46. Its complexity increases to $O(n^2)$. Note

Algorithm 15 FEASIBLE and SCHEDULE for Pure SOSP

```
1:  procedure FEASIBLE(S_i)
2:      C ← 0                                          ▷ Current time
3:      for (j, k, τ, ., .) in S_i do
4:          C ← max{C, r_j^k} + p_j^k
5:          if C > d_j^k then
6:              return false
7:      return true
8:  procedure SCHEDULE(S_i)
9:      C ← 0                                          ▷ Current time
10:     w ← 0                                          ▷ Total profit
11:     for (j, k, τ, s, p) in S_i do
12:         C ← max{C, r_j^k}
13:         s ← C
14:         p ← p_j^k
15:         w ← w + w_j
16:         C ← C + p
17:     return w
```

that it requires a priority queue. However, this priority queue is actually implemented with an array instead of a binary heap; since the number of observations in a night is less than 10, the array implementation happens to be faster.

For Pure df-SOSP, SNTP can be solved using Dynamic Programming based on the following recursive function:

$$w(j,t) = \max \begin{cases} w(j-1, \min\{t, d_j^i\} - p_j^i) + w_j & \text{if } \min\{t, d_j^i\} - p_j^i) \geq r_j^i \\ w(j-1, \min\{t, d_j^i\} - xp_j^i) + xw_j & \text{if } \min\{t, d_j^i\} - xp_j^i) \geq r_j^i \\ -\infty \end{cases}$$

The optimal value is then $w(|S_i|, d)$ where s is the deadline of the last observation of S_i. It is implemented by states and the solution is retrieved by storing the observations scheduled in an additional integer in each state: if bit $j = 0$, then the jth observation is scheduled during its minimum duration, otherwise, it is scheduled during its maximum duration (see Kellerer et al. 2004 for more details on efficient Dynamic Programming implementations)

Interestingly, SNTP for Pure SOSP is equivalent to SNTP for IPAG SOSP. Indeed, all the additional constraints do not change the sub-problem. Therefore, Algorithm 15 can be straightforwardly adapted. Likewise, SNTP for

Algorithm 16 SCHEDULE for Pure cf-SOSP

procedure SCHEDULE(S_i)
  $C \leftarrow 0$ ▷ Current time
  pos $\leftarrow 0$ ▷ Position of the current observation
  for (j, k, τ, s, p) in S_i **do**
    $C \leftarrow \max\{C, r^i_j\}$
    $s \leftarrow C$
    $p \leftarrow p^i_j$
    $C \leftarrow C + p$
    while $C > d^i_j$ **do**
      Let $\text{pos}_{\min}$ be the position of the first observation starting at its release date.
      Let (j', k', τ', s', p') be the observation at position pos$'$ of S_i such that $\text{pos}_{\min} \leq \text{pos}' \leq$ pos that maximizes $w_{j'}/p^i_{j'}$.
      Let $p_{\max}$ be the maximum duration at which observation at position pos$''$ can be reduced and all observations at position pos$''$, pos$' <$ pos$'' \leq$ pos can be shifted to the left without violating their release dates.
      $p' \leftarrow p' - p_{\max}$
      $C \leftarrow s' + p'$
      for pos$''$ from pos$' + 1$ to pos **do**
        $s'' \leftarrow C$
        $C \leftarrow C + p''$
    pos $\leftarrow$ pos $+ 1$
  $w \leftarrow 0$
  for (j, k, τ, s, p) in S_i **do**
    $w \leftarrow w + p/p^i_j * w_j$
  return w

IPAG df-SOSP can be solved by Dynamic Programming as explained for Pure df-SOSP.

II.3.2 Greedy algorithm

Algorithm 17 is a greedy algorithm that solves the full star observation scheduling problem. It uses SNTP and relies on the FEASIBLE function described in the previous section.

Algorithm 17 Greedy algorithm

procedure GREEDY(NightSet, TWList)
    ▷ time-windows from TWList all belong to nights of NightSet
    scheduled[j] $\leftarrow$ false, for all $j = 1, \ldots, n$
    $S_i = \emptyset$ for all $i \in$ NightSet
    for $\mathrm{TW}_{\mathrm{j}}^{\mathrm{k}}$ in TWList **do**
        if scheduled[j] **then**
            continue
        $\tau \leftarrow U\left(\min\left\{e_{jk}^{\min}, s_{jk}^{\max}\right\}, \max\left\{e_{jk}^{\min}, s_{jk}^{\max}\right\}\right)$
        Insert $(j, k, \tau, ., .)$ to $S_{m_{jk}}$
        if FEASIBLE($S_{m_{jk}}$) **then**
            scheduled[j] $\leftarrow$ true
        else
            Remove $(j, k, \tau, ., .)$ from $S_{m_{jk}}$
    return $\{S_i\}_{i \in \text{NightSet}}$

Algorithm 17 simply inserts time-windows in the order they are given. If the current solution becomes infeasible, the last added observation is removed. τ is drawn randomly with a uniform distribution between $\min\left\{e_{jk}^{\min}, s_{jk}^{\max}\right\}$ and $\max\left\{e_{jk}^{\min}, s_{jk}^{\max}\right\}$ where $e_{jk}^{\min}$ and $s_{jk}^{\max}$ are respectively the earliest possible end date and the lastest possible start date of target T_j in time-window TW_j^k. This choice of τ ensures that if the order of two observations can be deduced *a priori*, then the two values of τ will be sorted accordingly. For example, if a time-window has mandatory instants, then τ will be one of them. Therefore if two time-windows have distinct mandatory instants, which imposes the order, the corresponding observations will be scheduled in the correct one (if they have one common mandatory instant, then they are not compatible). If two time-windows $\mathrm{TW}_{j_1}^{k_1}$ and $\mathrm{TW}_{j_2}^{k_2}$ satisfy $s_{j_1 k_1}^{\max} \leq e_{j_2 k_2}^{\min}$, then T_{j_1} cannot be scheduled after T_{j_2} and values of τ will again be sorted accordingly.
Algorithm 17 may actually run on a limited number of nights defined by the input parameter NightSet, as well as it does not necessarily process all time-

windows, but only those contained in TWList. Note that it does not call the SCHEDULE function, as it may be significatively slower than FEASIBLE. This way, we ensure that Algorithm 17 runs very fast whatever the variant of problem.

II.3.3 Local search

Based of the Greedy algorithm described in the previous section, we developed a simple Large Neighorhood Search algorithm. Algorithm 18 generates a first solution by running the Greedy algorithm on the whole instance. Then the solution is improved by continuously rescheduling a set of nights $\bar{I}$ with the Greedy algorithm. The number of nights $|\bar{I}|$ is drawn randomly such that $P(|\bar{I}| = z) = 1/2^{z-1}$ with $z \geq 2$. Then $|\bar{I}|$ nights are drawn uniformly among all nights.
In the initial stage of the algorithm, time-windows are considered by efficiency.
The efficiency is a heuristic measure of the interest of a time-window. We use the ratio between the profit of the time-window and the typical processing-time. More formally, for Pure SOSP, Pure cf-SOSP and Pure df-SOSP, we choose:

$$\text{EFFICIENCY}(\text{TW}_j^k) = \frac{w_j}{p_j^k}$$

and for IPAG SOSP and IPAG df-SOSP, we choose:

$$\text{EFFICIENCY}(\text{TW}_j^k) = \frac{w_j^k}{p_j^k \left(\frac{s_{jk}^{\min} - e_{jk}^{\max}}{2} \right)}$$

For the improvment phase, we need to increase the diversity of the solution and to this extend we use a *disturbed* efficency: we add a random term, uniformly drawn in $[-0.001E_{\max}; 0.0001E_{\max}]$, where $E_{\max}$ is the maximum efficiency among all time-windows.
Finally, a simple mechanism is implemented to get out of local optima: a solution is accepted if its absolute gap with the best known solution is not greater than $y = 1.1$ (the profit of a full observation lies between 10 and 40).

II.4 Computational experiments

Catusse et al. 2016 performed their experiments on an Intel® Xeon E5-2440 v2 @ 1.9GHz. Due to the huge total computation time (several weeks), the new experiments were run on two different computers, but with no incidence

Algorithm 18 Large Neighorhood Search

procedure LNS(I)
  TWList $\leftarrow$ TW$_j^k$ for all $j = 1, \ldots, n$, $k = 1, \ldots, K_j$
  Sort TWList by efficiency
  $S \leftarrow$ GREEDY($\mathcal{M}$, TWList)
  $S^{\text{best}} \leftarrow S$
  while true **do**
    Draw a set of random nights $\bar{I}$
    Let TWList be the set of time-windows belonging to nights of $\bar{I}$ from unscheduled targets and targets scheduled in nights of $\bar{I}$
    Sort TWList by disturbed efficiency
    $S'_{\bar{I}} \leftarrow$ GREEDY $\left(\bar{I}, \text{TWList}\right)$
    $w = w(S) - w(S_{\bar{I}}) + w(S'_{\bar{I}})$
    if $w \geq w(S^{\text{best}}) - y$ **then**
      $S_{\bar{I}} \leftarrow S'_{\bar{I}}$
      if $w_{\max} < w$ **then**
        $S^{\text{best}} \leftarrow S$

on the comparisons. Experiments on Pure SOSP, Pure df-SOSP and Pure cf-SOSP have been performed on an Intel® Core™ i5-8500 CPU @ 3.00GHz × 6. Experiments on IPAG SOSP and IPAG df-SOSP have been performed on an Intel® Core™ i5-6300U CPU @ 2.40GHz × 4.

II.4.1 Pure Star Observation scheduling problem

We used the 21 instances from Catusse et al. 2016 (a real instance and 20 generated ones), with time limits of 2 minutes (120 seconds) and 2 hours (7200 seconds). The best known upper bounds (UB) are the solutions provided by the Branch-and-Price from the same article.
Table II.1 compares the result of the Local Search from Catusse et al. 2016 (LS) with the proposed Large Neighorhood Search (LNS). Note that the comparison is not completely fair since the processor used for LNS is much faster. However, LNS clearly outperforms LS as it returns better solutions in 2 minutes than LS in 2 hours.
LNS usually fails to find an optimal solution (last column), but nevertheless provides very quickly good solutions and quickly very good ones. Indeed, Figure II.1 depicts the maximum relative gap between LNS and the best known upper bound on Pure SOSP for each size of instances. It gives an experimental guarantee of the solution quality for the user. At each instant, the largest gap always comes from an instance of size 1000. In 2 seconds,

Ins.	n	m	UB	LS 120	LS 7200	LNS 120	LNS 7200	LNS 120 - LS 120	LNS 120 - LS 7200	LNS 7200 - LS 7200	UB - LNS 7200
1	400	71	9 810	9 720	9 770	9 770	9 800	50	0	30	10
2	400	71	9 420	9 350	9 390	9 380	9 400	30	-10	10	20
3	400	71	9 920	9 880	9 910	9 910	9 920	30	0	10	0
4	400	71	9 300	9 230	9 250	9 290	9 300	60	40	50	0
5	400	71	9 260	9 180	9 200	9 220	9 250	40	20	50	10
6	600	107	14 340	14 160	14 260	14 280	14 300	120	20	40	40
7	600	107	14 120	13 910	14 000	14 050	14 070	140	50	70	50
8	600	107	14 740	14 560	14 660	14 700	14 720	140	40	60	20
9	600	107	14 020	13 890	13 840	13 970	13 980	80	130	140	40
10	600	107	13 600	13 400	13 450	13 540	13 560	140	90	110	40
11	800	142	19 450	19 270	19 350	19 370	19 410	100	20	60	40
12	800	142	19 630	19 350	19 450	19 510	19 530	160	60	80	100
13	800	142	19 940	19 740	19 850	19 850	19 880	110	0	30	60
14	800	142	19 110	18 900	19 000	19 020	19 050	120	20	50	60
15	800	142	18 770	18 560	18 660	18 650	18 690	90	-10	30	80
16	1 000	142	23 410	22 890	23 190	23 270	23 320	380	80	130	90
17	1 000	142	23 390	22 960	23 120	23 200	23 280	240	80	160	110
18	1 000	142	23 710	23 220	23 420	23 470	23 570	250	50	150	140
19	1 000	142	23 020	22 530	22 790	22 800	22 870	270	10	80	150
20	1 000	142	22 710	22 250	22 440	22 520	22 580	270	80	140	130
real	800	142	18 620	18 430	18 590	18 550	18 580	120	-40	-10	40

Table II.1 – Performances of LNS after 2 minutes (LNS120) and 2 hours (LNS7200) compared to best-known upper bounds (UB) and local search from Catusse et al. 2016 (LS120, LS7200).

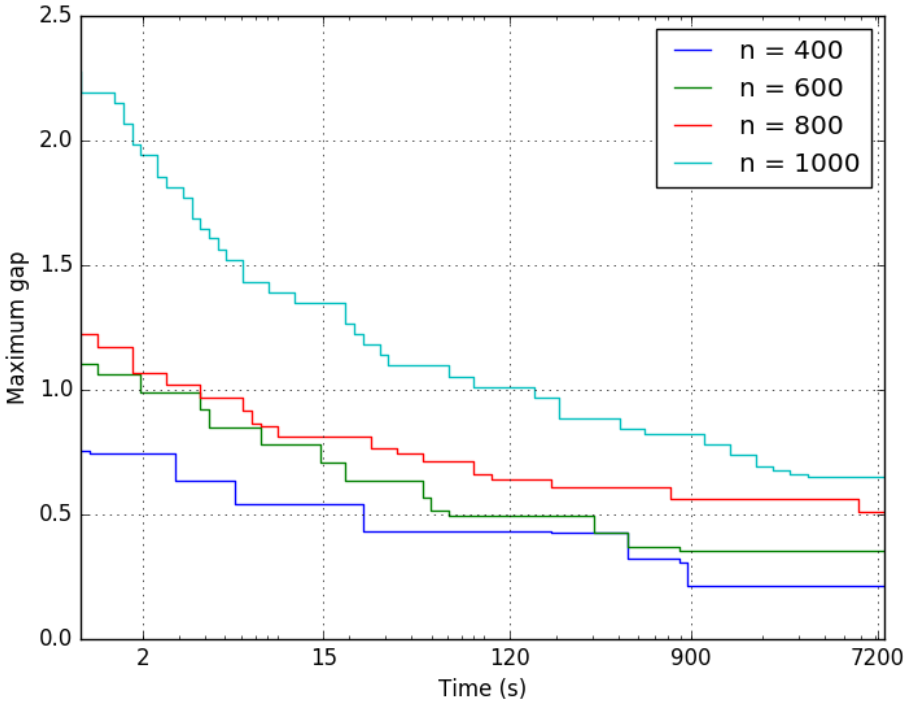


Figure II.1 – Maximum gap of LNS on Pure SOSP for each size of instances

the worst gap is below 2% and in 2 minutes, it is below 1%. Instances of size 1000 have been generated to be larger than real instances. If we only consider instances of size 400, 600 and 800, then the gap goes below 1% in a dozen of seconds. Therefore, LNS provides very good industrial solutions in a very short time.
From a theoretical point of view, we note that even if LNS keeps improving solutions all along the search, it is not able to provide optimal solutions within the time the Branch-and-price from Catusse et al. 2016 took to find them. Therefore, further research could focus on improving LNS on long runs.

II.4.2 Pure Flexible Star Observation scheduling problems

In this section, we try to assess the effect of including flexible observation duration in the model. We used the same instances as in the previous section. Our interests are: do we get better solutions with LNS when allow-

Ins.	UB	LNS 120	LNS 7200	LNS df 120	- LNS 120	LNS df 7200	- LNS 7200	- UB	LNS cf 7200	- LNS df 7200
1	9 810	9 770	9 800	9 826.50	56.5	9 837	37	27	9 835.90	-1.10
2	9 420	9 380	9 400	9 410.00	30	9 418	17.5	-2.5	9 426.82	9.32
3	9 920	9 910	9 920	9 945.50	35.5	9 967	47	47	9 968.89	1.89
4	9 300	9 290	9 300	9 317.50	27.5	9 325	24.5	24.5	9 322.72	-1.78
5	9 260	9 220	9 250	9 263.00	43	9 272	22	12	9 270.10	-1.90
6	14 340	14 280	14 300	14 333.00	53	14 354	53.5	13.5	14 347.44	-6.06
7	14 120	14 050	14 070	14 104.50	54.5	14 116	46	-4	14 108.33	-7.67
8	14 740	14 700	14 720	14 733.50	33.5	14 744	24	4	14 753.12	9.12
9	14 020	13 970	13 980	14 025.00	55	14 035	55	15	14 041.69	6.69
10	13 600	13 540	13 560	13 596.50	56.5	13 607	47	7	13 605.64	-1.36
11	19 450	19 370	19 410	19 429.50	59.5	19 450	39.5	-0.5	19 449.49	-0.01
12	19 630	19 510	19 530	19 569.00	59	19 589	59	-41	19 588.99	-0.01
13	19 940	19 850	19 880	19 890.00	40	19 910	30	-30	19 900.00	-10.00
14	19 110	19 020	19 050	19 070.00	50	19 080	30	-30	19 080.00	0.00
15	18 770	18 650	18 690	18 700.00	50	18 720	30	-50	18 730.00	10.00
16	23 410	23 270	23 320	23 358.00	88	23 403	82.5	-7.5	23 390.73	-11.77
17	23 390	23 200	23 280	23 313.50	113.5	23 367	86.5	-23.5	23 343.90	-22.60
18	23 710	23 470	23 570	23 569.00	99	23 630	60	-80	23 630.00	0.00
19	23 020	22 800	22 870	22 885.50	85.5	22 950	80	-70	22 939.74	-10.26
20	22 710	22 520	22 580	22 594.50	74.5	22 650	70	-60	22 650.00	0.00
real	18 620	18 550	18 580	18 656.00	106	18 678	98	58	18 677.10	-0.90

Table II.2 – Comparison of the values of the solutions returned by LNS on Pure SOSP, Pure df-SOSP and Pure cf-SOSP.

ing continous or discrete flexible observation durations? In this case, is the value of these solutions greater than the best known solutions for the original problem? do we get better results when possible observation durations are continuous? and how does the value of x influence the quality of the solutions?
Table II.2 compares the results obtained with LNS on Pure SOSP, Pure df-SOSP and Pure cf-SOSP with $x = 0.95$.
First we note that LNS always returns better solutions on Pure df-SOSP compared to Pure SOSP within 2 minutes (column 6, "- LNS 120") and 2 hours (column 8, "- LNS 7200"). Allowing observation duration flexibility indeed improves the solution value. However, the additional profit is rather moderate, on average 50 (0.3%) and a max of 100 (0.5%) on the instances of the dataset (let us remind that the profit of an observation lies between 10 and 40).
Then, from a more theoretical point of view, we note that on 9 instances, the value found by LNS on Pure df-SOSP is greater than the best-known upper bound for Pure SOSP (column 9, "- UB").
We also compare the solutions obtained on the discrete and the continuous models. Even though solutions of Pure df-SOSP are all feasible solutions of cf-SOSP, the values returned by the algorithm are equivalent for both models. Indeed, the mean difference between the discrete and the continous solution values is 1.83 with a standard deviation of 5.57 (for comparison, the mean difference between the discrete version and the non-flexible version is 49.48 with a standard deviation of 18.97). Since the discrete version is simpler, it should be prefered.
Finally, Figure II.2 shows the value returned by LNS for several values of x on two typical instances. The more the duration can be reduced, the greater the additional profit is, until $x = 0.85$. Then, the profit does not increase anymore.

II.4.3 IPAG Star Observation Scheduling Problems

On the more realistic version of the problem, we tested our LNS algorithm on four new real instances provided by the astrophysicists.
To this extend, LNS was integrated within SPOT, the software developed for the astrophysicists, and compared to the Simulated Annealing algorithm (SA) they currently use. Figure II.3 shows the schedule returned by LNS for one of them.
Since there are only four instances, we directly show the graphs for each one. Figure II.4 and II.5 depict the profit at each instant for SA (on IPAG SOSP), LNS on IPAG SOSP and LNS on IPAG df-SOSP. LNS is able to find better

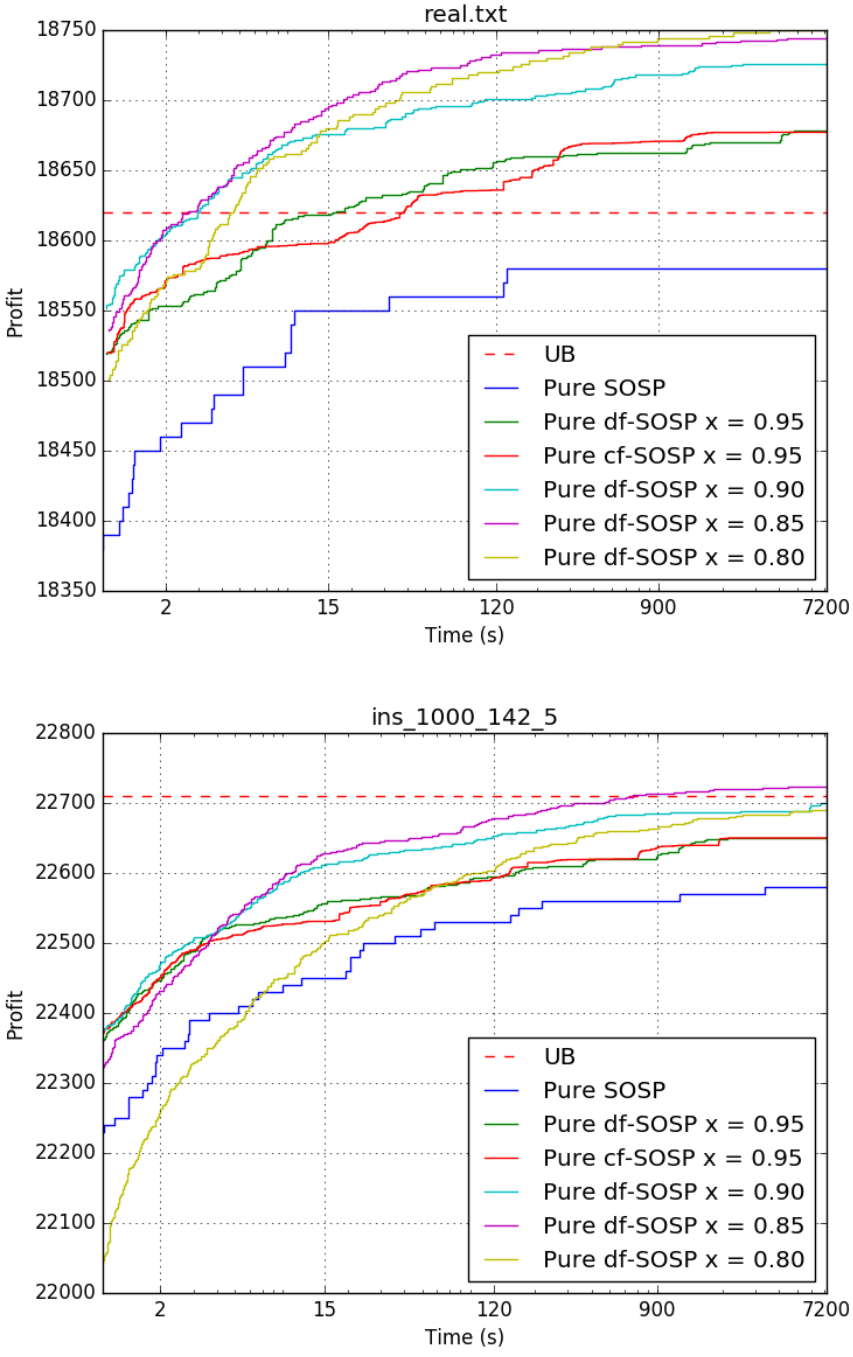


Figure II.2 – Comparison between the values of the solutions returned by LNS for several values of x on two typical instances.

solutions than SA. Furthermore, it finds them much quicker. When flexible observation duration is allowed, we observe the same behaviour as for Pure df-SOSP: the total profit is slightly greater.

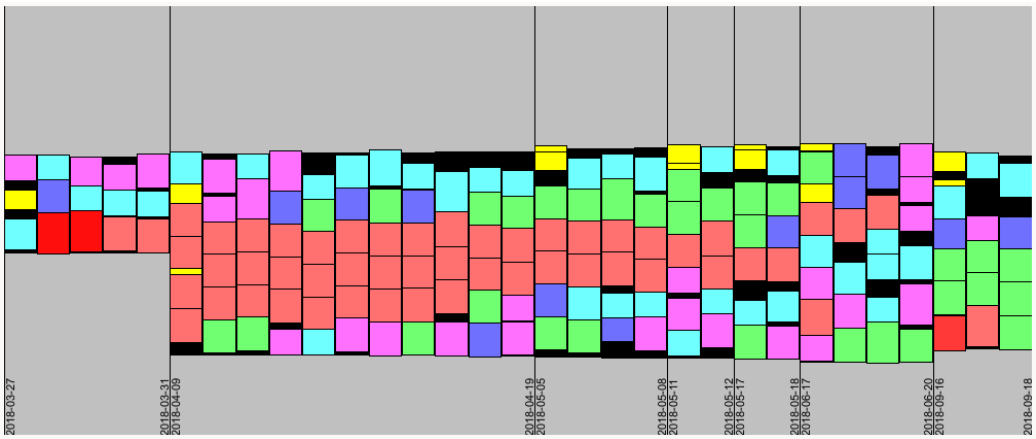


Figure II.3 – Visualisation of a schedule found by LNS in SPOT. Each column corresponds to a night, each rectangle to an observation. The height of a rectangle corresponds to the duration and its color to the profit of the corresponding observation.

II.5 Conclusion

We proposed a simple and general Large Neighborhood Search scheme for star observation scheduling problems. The algorithm reduced the resolution of a specific variant to the resolution of a Single Night Timing Problem for which algorithms are much easier to develop and adapt. We illustrated this by implementing several algorithms solving SNTP for different variants of star observation scheduling problems.
Our Large Neighborhood Search algorithm is relevant because it clearly outperforms both the previous Local Search algorithm on literature instances for the pure problem, and the currently used SA algorithms on real instances. Besides, it is simpler and more easily adaptable than both.
Compared to the Branch-and-Price algorithm, it provides solutions within 2% of optimality in less than 2 seconds, while the Branch-and-Price will need about 10 minutes to find its first solution on real sized instances. However, even if the Large Neighborhood Search continues improving the solution over time, the solution found by the Branch-and-Price is almost always the optimal one and the Large Neighborhood Search cannot find it within the same

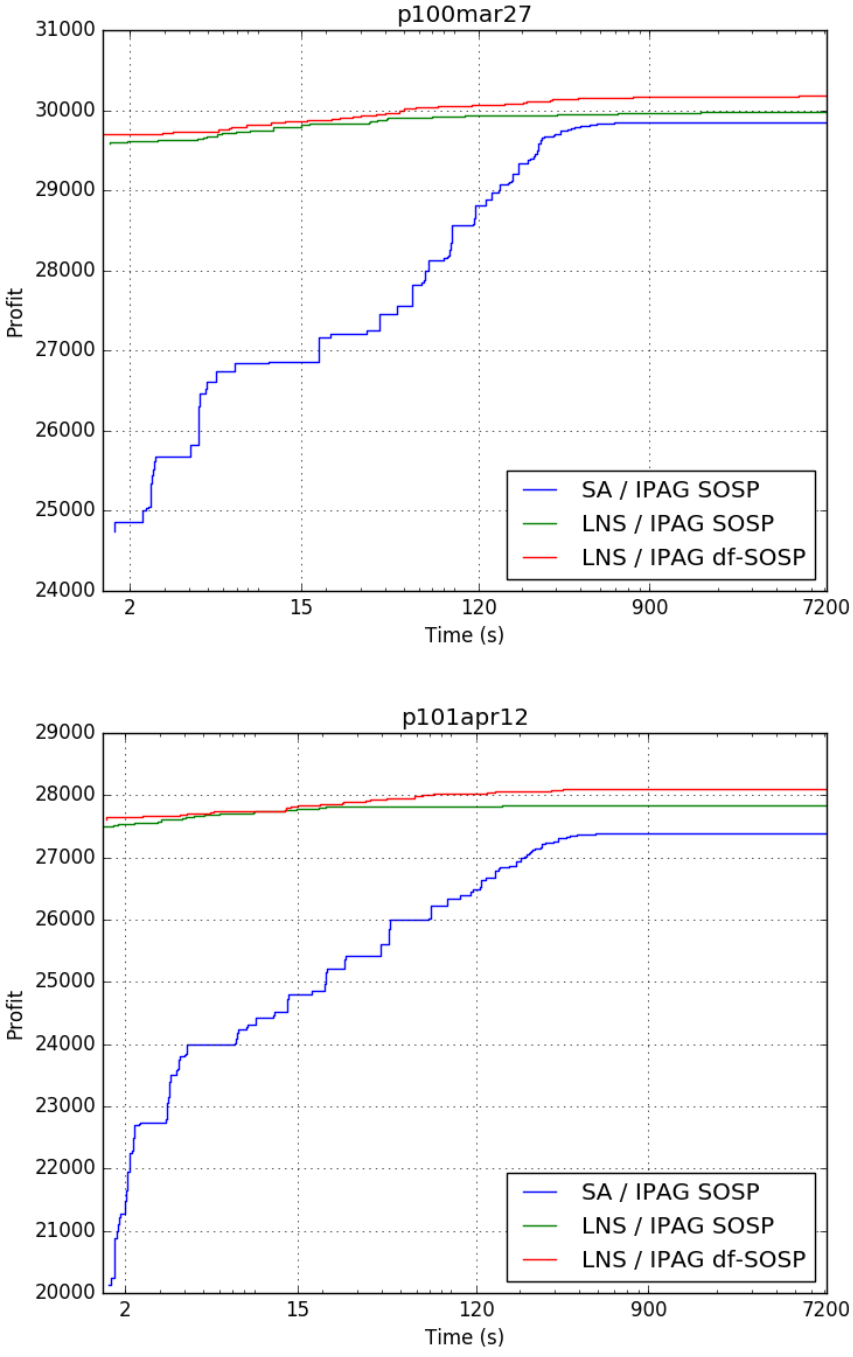


Figure II.4 – Comparison between the values of the solutions returned by the algorithms implemented in SPOT on instances `p100mar27` and `p101apr12`.

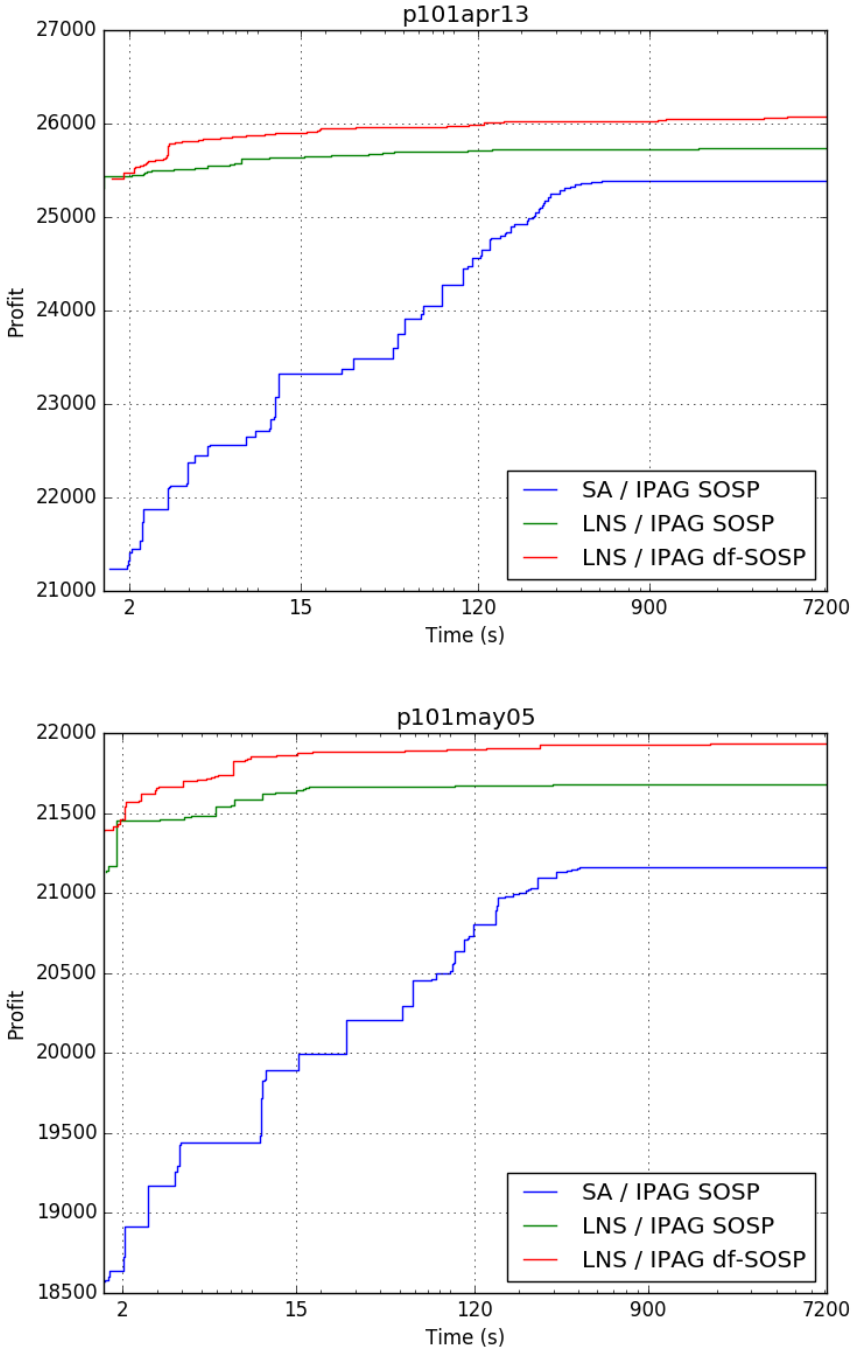


Figure II.5 – Comparison between the values of the solutions returned by the algorithms implemented in SPOT on instances `p100apr13` and `p101may05`.

amount of time. Further research could for example focus on diversification techniques to improve this aspect.
Theoretically, allowing flexible observation durations necessarily improves the value of the optimal solution. In practice, on the given instances, LNS indeed finds better solutions in theses cases. However, the additional gain is rather moderate.
The algorithm is now implemented in the software used by the astrophysicists, and we hope it will be used in production very soon.

References

Bianchessi, Cordeau, Desrosiers, Laporte, and Raymond (2007). "A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites". In: *European Journal of Operational Research* 177.2, pp. 750–762. DOI: 10.1016/j.ejor.2005.12.026.

Catusse, Cambazard, Brauner, Lemaire, Penz, Lagrange, and Rubini (2016). "A Branch-And-Price Algorithm for Scheduling Observations on a Telescope". In: *Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI Press, pp. 3060–3066.

Dilkina and Havens (2005). "Agile satellite scheduling via permutation search with constraint propagation". In: *ed* 12.

Hall and Magazine (1994). "Maximizing the value of a space mission". In: *European Journal of Operational Research*. Project Management and Scheduling 78.2, pp. 224–241. DOI: 10.1016/0377-2217(94)90385-9.

He, Liu, Laporte, Chen, and Chen (2018). "An improved adaptive large neighborhood search algorithm for multiple agile satellites scheduling". In: *Computers & Operations Research* 100, pp. 12–25. DOI: 10.1016/j.cor.2018.06.020.

Janiak and Kovalyov (1996). "Single machine scheduling subject to deadlines and resource dependent processing times". In: *European Journal of Operational Research* 94.2, pp. 284–291. DOI: 10.1016/0377-2217(96)00129-4.

Johnston (1990). "Spike: AI scheduling for NASA's Hubble Space Telescope". In: *Sixth Conference on Artificial Intelligence for Applications*, 184–190 vol.1. DOI: 10.1109/CAIA.1990.89188.

Johnston (1993). "The Application of Artificial Intelligence to Astronomical Scheduling Problems". In: *Astronomical Data Analysis Software and Systems II*. Vol. 52, p. 329.

Kellerer, Pferschy, and Pisinger (2004). *Knapsack Problems*. en. Berlin, Heidelberg: Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-24777-7.

Lagrange, Rubini, Brauner, Cambazard, Catusse, Lemaire, and Baude (2016). "SPOT: an optimization software for dynamic observation programming". In: *SPIE 9910, Observatory Operations: Strategies, Processes, and Systems VI, 991033 (July 18, 2016)*. Edinburgh, United Kingdom. DOI: 10.1117/12.2241197..

Lemaître, Verfaillie, Jouhaud, Lachiver, and Bataille (2002). "Selecting and scheduling observations of agile satellites". In: *Aerospace Science and Technology* 6.5, pp. 367–381. DOI: 10.1016/S1270-9638(02)01173-2.

Li, Xu, and Wang (2007). "Scheduling Observations of Agile Satellites with Combined Genetic Algorithm". In: *Third International Conference on Natural Computation (ICNC 2007)*. Vol. 3, pp. 29–33. DOI: 10.1109/ICNC.2007.652.

Peng, Dewil, Verbeeck, Gunawan, Xing, and Vansteenwegen (2019). "Agile Earth Observation Satellite Scheduling: an Orienteering Problem with Time-Dependent Profits and Travel Times". In: *Computers & Operations Research*. DOI: 10.1016/j.cor.2019.05.030.

Pisinger and Ropke (2010). "Large neighborhood search". In: *Handbook of metaheuristics*. Springer, pp. 399–419.

Verfaillie and Lemaître (2001). "Selecting and Scheduling Observations for Agile Satellites: Some Lessons from the Constraint Reasoning Community Point of View". en. In: *Principles and Practice of Constraint Programming — CP 2001*. Ed. by Toby Walsh. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 670–684.

Wang, Reinelt, Gao, and Tan (2011). "A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation". In: *Computers & Industrial Engineering*. Combinatorial Optimizatiion in Industrial Engineering 61.2, pp. 322–335. DOI: 10.1016/j.cie.2011.02.015.

Wolfe and Sorensen (2000). "Three Scheduling Algorithms Applied to the Earth Observing Systems Domain". In: *Management Science* 46.1, pp. 148–166. DOI: 10.1287/mnsc.46.1.148.15134.

Xu, Chen, Liang, and Wang (2016). "Priority-based constructive algorithms for scheduling agile earth observation satellites with total priority maximization". In: *Expert Systems with Applications* 51, pp. 195–206. DOI: 10.1016/j.eswa.2015.12.039.

Chapter III

ROADEF/EURO challenge 2018

done and written with Luc Libralesso

Every other year, the French Operational Research and Decision Support Society (ROADEF) jointly with the European Operational Research Society (EURO) organizes an optimization challenge open to everyone. Each edition is a collaboration with an industrial partner, and the problems and instances submitted correspond to real life industrial optimization problems. Previous challenges have been organized in collaboration with Air Liquide (2016), SNCF (2014), Google (2012), EDF (2010)...
The 2018 edition of the challenge was dedicated to a cutting optimization problem in collaboration with Saint-Gobain[1]. The challenge was announced in February 2018. The sprint phase ended in June 2018, the qualification phase in September 2018 and the final phase in January 2019. We took part in the challenge shortly before the end of the qualification phase.
In this chapter, we present the heuristic *Branch and Bound* algorithm we submitted for the final phase of the challenge. The resulting program was ranked first over 20 qualified submissions. The code is free (GPL-3.0) and available online[2].

In the AI community, solving problems often involves exploring a search tree, either exhaustively, using methods similar to *Branch and Bound*, or using heuristic procedures that only explore the most promising parts of the search tree. In the latter case, they are called *Anytime Tree Searches* because they can be stopped at any time and provide good solutions in the allowed computation time. Such methods have the same objective as metaheuristics. Anytime Tree Searches usually start by the most promising regions of the search tree, providing good solutions first. Then they explore the least promising regions if time permits it. Possibly, the tree search will explore the whole tree and thus it will provide the optimal solution. To the best

[1] http://www.roadef.org/challenge/2018/en/
[2] https://github.com/fontanf/roadef2018

of our knowledge, *Anytime Tree Searches* are hardly present in *Operations Research*.

For the ROADEF/EURO challenge 2018, we took the bet that one can combine *Branch and Bounds* from Operations Research and Tree Search Algorithms from AI to get a simple method that is competitive with classical meta-heuristics. We designed a Branch and Bound, with pseudo-dominance properties, symmetry breaking rules, bounds and heuristic guides as people usually do in Operations Research. We changed the search strategy by a custom iterative tree search algorithm inspired from AI. It starts its first iteration by performing very aggressive heuristic cuts and behaves like a greedy algorithm finding a good solution fast. At the second iteration, it performs less aggressive heuristic cuts, taking more time, finding even better solutions. If the algorithm runs for long enough, no heuristic cut will be performed and it behaves like an exact Branch and Bound. The resulting method obtained the best results compared to the other submitted approaches during the final phase.

We also extracted a general framework that can be applied to many other industrial problems. Indeed, the method can be divided into two parts. First, one designs the *Branching Scheme* which is a definition of the implicit search tree (*i.e.* root node, how to generate children of a given node, lower bounds *etc.*). This implicit search tree is usually too big to be entirely explored. Thus, we also need to design a strategy to explore the tree starting from promising regions. This decomposition allows rapid prototyping of both search trees definitions and tree searches. We believe that many tree search techniques from AI can be applied with success on many Operations Research problems.

The chapter is organized as follows: Section III.1 contains the problem description; Section III.2 describes the branching scheme (*i.e.* implicit search tree definition) that we implemented; Section III.3 presents an overview of the most significant *tree search* algorithms in AI and the algorithm *Memory Bounded A** (MBA*) that we designed for the challenge. Eventually, Section III.5 is dedicated to computational experiments on the challenge instances.

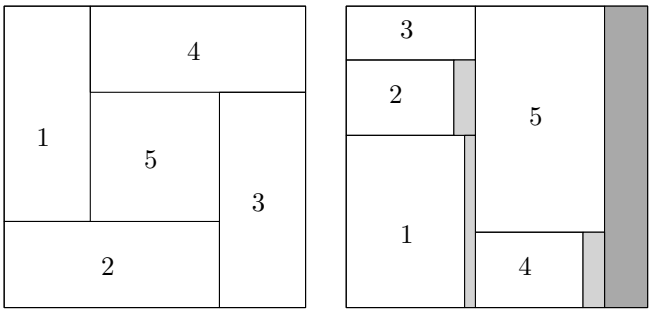
III.1 Problem definition

The ROADEF/EURO challenge 2018 was proposed by the company *Saint-Gobain*. The challenge consists in packing rectangular glass items into standardized bins called jumbos. Each jumbo has a fixed size $H \times W$ (6m $\times$ 3.21m). The goal is to produce all items while minimizing the total surface

of waste (surface not used by items). For the last jumbo used, we allow a special cut creating a remaining part that will not be counted as waste. A more detailed description of the problem and examples can be found on the challenge website[3].
Formally, the objective can be formulated as:

$$\min \left(nHW - Hw - \sum_{i \in \mathcal{I}} w_i h_i \right)$$

where n is the number of jumbos used; W and H the standardized width and height of jumbos; w the position of the most right cut of the last jumbo; and $\mathcal{I}$ the set of produced items of height h_i and weight w_i, $i \in \mathcal{I}$. Note that the last term is constant but necessary to interpret the objective as the total waste of the solution.



(a) Non valid guillotine cut (b) valid guillotine cut

Figure III.1 – Illustration of a not valid guillotine cut (a) and a valid one (b). Fig (b) shows in light gray possible waste areas and in dark gray the possible remaining area that can serve other uses.

The problem is a variant of the classical two-dimensional orthogonal (rectangular items and cuts parallel to the border of the jumbo) packing problem. However, it contains several additional constraints that makes it more difficult to solve:

- Cuts must be guillotine (i.e. traversing all the jumbo). Figure III.1 shows an example of a non-guillotine solution and a guillotine one. The vertical cuts traversing the whole jumbo are called 1-cuts. Each horizontal cut separating regions within a 1-cut is called a 2-cut. The same principle applies for 3-cuts and 4-cuts.

[3] http://www.roadef.org/challenge/2018/en/index.php

- Items are subject to chain precedence constraints. Each item belongs to exactly one chain. Some instances have a few chains, some have no precedence constraint (*i.e.* there are $|I|$ chains). When extracted from left to right and bottom to top, the items in a chain need to be in the right order.

- Jumbos can contain defects (between 0 to 8 rectangles about 1-2cm wide). For quality reasons, those defects should not be present in items and cuts cannot be performed on such defects. One needs to pack those defects in the waste.

- Only 4 levels of cuts are available. Moreover, it is possible to perform at most one 4-cut in a sub-plate obtained after a 3-cut. Thus, after a 3-cut, the sub-plate can contain waste, one item without waste, one item and some waste or two items without waste. This is illustrated on Figure III.2.

- Cuts, depending on their level are subject to minimum and maximum size constraint. The distance between two consecutive 1-cuts must be between $w^1_{\min} = 100$ and $w^1_{\max} = 3500$. The distance between two consecutive 2-cuts must be at least $w^2_{\min} = 100$. Finally, the width and the height of any waste area must be at least $w_{\min} = 20$. The consequence of this last constraint is not straightforward and is illustrated on Figure III.3.

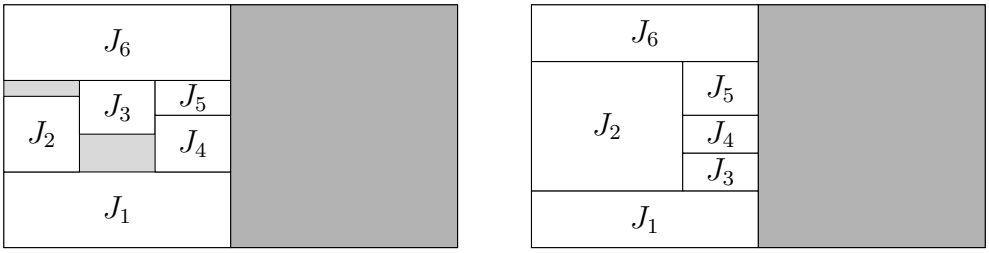


Figure III.2 – Only one 4-cut is allowed. Therefore, the first solution is feasible but the second is not.

III.2 Branching scheme

In this section, we describe the branching scheme used by our Tree Search Algorithms. The first part provides several theoretical dominance properties

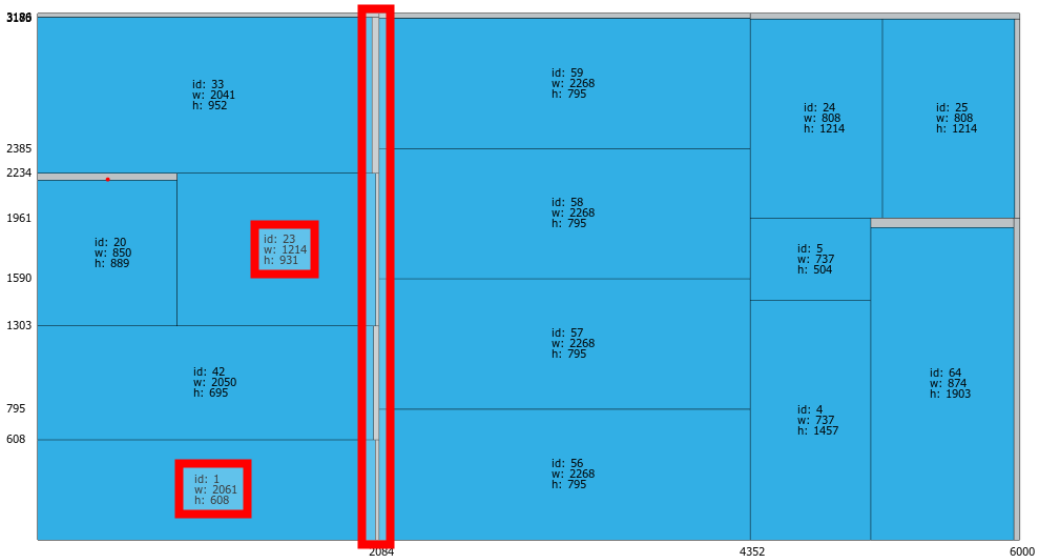


Figure III.3 – Illustration of the consequence of the minimum waste constraint on a plate of the best known solution of instance $B1$: additional waste must be added before the first 1-cut. Otherwise either the waste on the right of item 1 or the waste on the right of item 23 would violate the minimum waste constraint.

that justify the heuristic choices that we made. Then we describe the general scheme. Finally, we describe a pseudo-dominance rule and a symmetry breaking strategy.

In this section, S denotes a solution (items and cuts with their positions) or a partial solution (not all items may be placed), J an item (or a defect) and D a defect.

For all item $J \in S$, $x_1(J,S)$, $x_2(J,S)$, $y_1(J,S)$ and $y_2(J,S)$ denote respectively the positions of its left cut, right cut, bottom cut and top cut. S is omitted when there is no ambiguity. For a defect D, $x_1(D)$, $x_2(D)$, $y_1(D)$ and $y_2(D)$ denote respectively its left border, right border, bottom border and top border (that do not depend on a solution). For a k-cut $C \in S$, k odd, $x(C)$ denotes its x coordinate and $y_1(C) < y_2(C)$ denote its starting and ending positions. For a k-cut $C \in S$, k even, $y(C)$ denotes its y coordinate and $x_1(C) < x_2(C)$ denote its starting and ending positions.

Let S be a partial solution. Let $n(S)$ denote the number of plates used by S and for $k \in \{1,2,3\}$, let $x_k^{\text{curr}}(S)$ (resp. x_k^{prec}) be the position of the last k-cut (resp. the previous k-cut) of the last plate of S. If there is only one 1-cut on the last plate, then $x_1^{\text{prec}} = 0$. And if, for $k \in \{2,3\}$, there is only one k-cut between the x_{k-1}^{prec} and x_{k-1}^{curr}, then $x_k^{\text{prec}} = 0$. Current and previous

cuts are illustrated on Figure III.4.

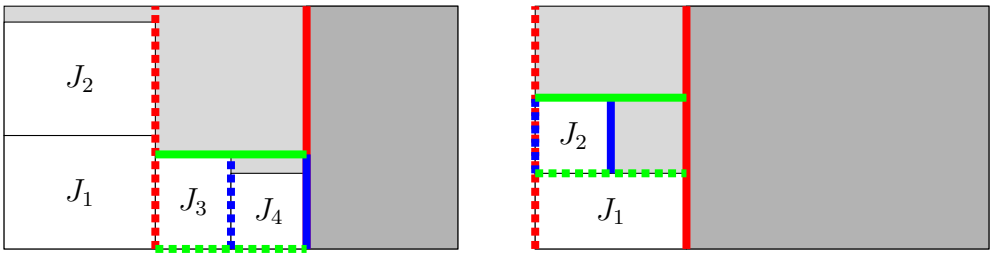


Figure III.4 – Examples of current (plain) and previous (dotted) cuts

III.2.1 Dominance results

Ideally, a branching scheme should make any solution reachable while limiting the number of equivalent or dominated ones. For this problem, it is not straightforward. Therefore, in this section, we derive several dominance properties to help us decide which branching scheme to implement.
In this section, "as late as possible" for cuts refers to the order in which they are performed.
First note that the border of an item corresponds to one unique cut, as illustrated on Figure III.5.

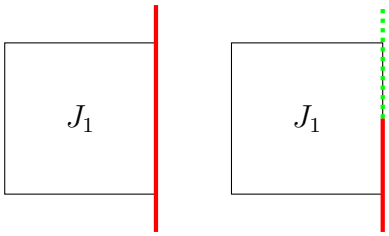


Figure III.5 – The second pattern cannot be obtained with guillotine cuts.

Therefore, if $C \in S$ is a k-cut, k odd, and J an item of S or a defect such that $x_2(J) = x(C)$. Then either $y_1(J) \geq y_2(C)$, or $y_2(J) \leq y_1(C)$, or $y_1(C) \leq y_1(J)$ and $y_2(J) \leq y_2(C)$. The equivalent holds if k is even.
In the pure two-dimensional orthogonal packing problem, it is dominant to pack the items in a corner. When adding some constraints of the current packing problem, this remains true:

Proposition III.2.1.1
For the guillotine two-dimensional orthogonal packing problem with defects and precedences, the set of solutions $\mathcal{U}$ such that for all $S \in \mathcal{U}$:

- for all k-cut C, k odd, there exists either an item or a defect J such that $y_1(J) \geq y_1(C)$ and $y_2(J) \leq y_2(C)$ and $x_2(J) = x(C)$
- for all k-cut C, k even, there exists either an item or a defect J such that $x_1(J) \geq x_1(C)$ and $x_2(J) \leq x_2(C)$ and $y_2(J) = y(C)$

is dominant.

Proof. Let S be an optimal solution violating one of the conditions as late as possible. We prove the odd case, however the proof is identical for the even case. Let C be the first k-cut of S, k odd, such that for all items and defects J: $y_1(J) > y_2(C)$ or $y_2(J) < y_1(C)$ or $x_2(J) \neq x(C)$. C can be moved to the left with all items of S which left cut is C until it encounters an item, a defect, another cut, or the border of the plate (in the last two cases, the cut C is removed). The new solution remains optimal and either it violates the property *later* than S or not at all; in both cases, that contradicts the choice of S. ■

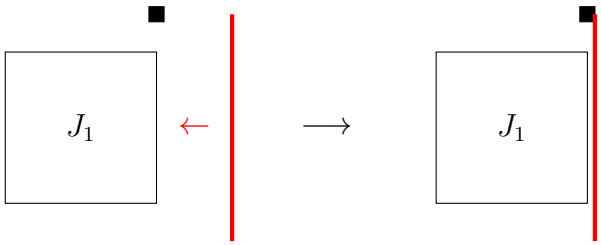


Figure III.6 – Illustration of Proof of Proposition III.2.1.1. The new solution is feasible since there is no minimum waste constraint.

The property holds with the minimum waste constraint if we remove the precedences and the defects:

Lemma III.2.1.1
For the guillotine two-dimensional orthogonal packing problem with minimum waste, the set of solutions $\mathcal{U}'$ such that for all $S \in \mathcal{U}'$, for all consecutive k-cuts C_1, C_2 (that may be a plate border), there exists at most one $(k+1)$-cut C that does not satisfy the following properties

- if $k+1$ is even, there exists either an item J such that $x_1(C) \leq x_1(J) \leq x_2(J) \leq x_2(C)$ and $y_2(J) = y(C)$, or two items J_1 and J_2 such that $x_1(C) \leq x_1(J_1) \leq x_2(J_1) \leq x_2(C)$, $x_1(C) \leq x_1(J_2) \leq x_2(J_2) \leq x_2(C)$, $y_2(J_1) = y(C) - w_{\min}$ and $y(C) - 2w_{\min} < y_2(J_2) < y(C) - w_{\min}$.

- if $k+1$ is odd, there exists either an item J such that $y_1(C) \leq y_1(J) \leq y_2(J) \leq y_2(C)$ and $x_2(J) = x(C)$, or two items J_1 and J_2 such that $y_1(C) \leq y_1(J_1) \leq y_2(J_1) \leq y_2(C)$, $y_1(C) \leq y_1(J_2) \leq y_2(J_2) \leq y_2(C)$, $x_2(J_1) = x(C) - w_{\min}$ and $x(C) - 2w_{\min} < x_2(J_2) < x(C) - w_{\min}$.

is dominant.

Proof. Let S be an optimal solution where the condition is not satisfied as late as possible. Let C and C' be the first pair of cuts violating the conditions, *i.e.* the first item or border before the cut is at strictly more than $w_{\min}$. C can be shifted backward as well as all the items placed between C and C' until C satisfies the condition (Figure III.7). The new solution is feasible and either it belongs to $\mathcal{U}'$, or it violates the condition later than S. Hence a contradiction. ■

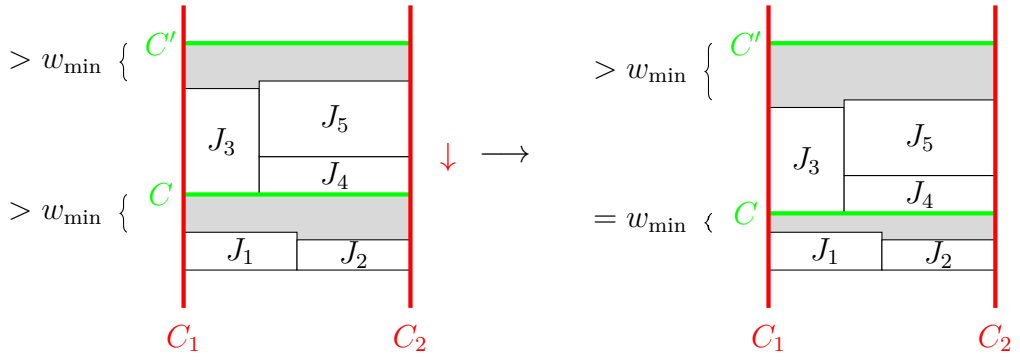


Figure III.7 – Illustration of Proof of Lemma III.2.1.1. The new solution is feasible since there are no defects.

Proposition III.2.1.2

For the guillotine two-dimensional orthogonal packing problem with minimum waste, the set of solutions $\mathcal{U}_2$ such that for all $S \in \mathcal{U}_2$:

- for all k-cut C, k odd, there exists either an item J such that $y_1(C) \leq y_1(J) \leq y_2(J) \leq y_2(C)$ and $x_2(J) = x(C)$, or two items J_1 and J_2 such that $y_1(C) \leq y_1(J_1) \leq y_2(J_1) \leq y_2(C)$, $y_1(C) \leq y_1(J_2) \leq y_2(J_2) \leq y_2(C)$, $x_2(J_1) = x(C) - w_{\min}$ and $x(C) - 2w_{\min} < x_2(J_2) < x(C) - w_{\min}$.
- for all k-cut C, k even, there exists either an item J such that $x_1(C) \leq x_1(J) \leq x_2(J) \leq x_2(C)$ and $y_2(J) = y(C)$, or two items J_1 and J_2 such that $x_1(C) \leq x_1(J_1) \leq x_2(J_1) \leq x_2(C)$, $x_1(C) \leq$

$x_1(J_2) \leq x_2(J_2) \leq x_2(C)$, $y_2(J_1) = y(C) - w_{\min}$ and $y(C) - 2w_{\min} < y_2(J_2) < y(C) - w_{\min}$.

is dominant.

Proof. Let $S \in \mathcal{U}'$ be an optimal solution such that it contains a k-cut C violating the condition (*i.e.* the first item or border before the cut is at strictly more than $w_{\min}$), $k = 1$ or there is no $(k-1)$-cuts violating the condition, and C is the first such cut. Let C_1 and C_2 be the previous and the next $(k-2)$-cuts or border of C. The blocks between C_1 and C and between C and C_2 can be exchanged. The new solution is feasible and the extra waste is now either before a border or before a $(k-2)$-cut. Hence a contradiction. ■

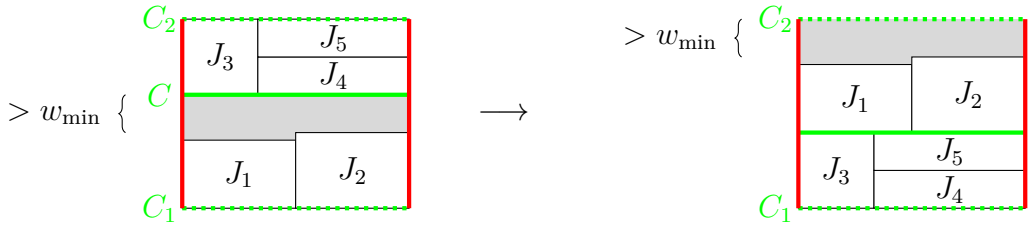


Figure III.8 – Illustration of Proof of Proposition III.2.1.2. The new solution is feasible since there are neither defects nor precedences.

However, with the minimum waste constraint and precedences, or with the minimum waste constraint and defects, the property does not hold. Counter-examples are illustrated on Figure III.9. Both instances contain three items J_1, J_2 and J_3. In the first instance, J_1 precedes J_3. In the second instance, there is a defect on the plate. Item J_1 is higher than item J_2 by strictly less than the minimum waste. This imposes the next 2-cut to be placed after $y_2(J_1) + w_{\min}$. However if this cut is placed exactly at $y_2(J_1) + w_{\min}$, then the distance between the top of item J_3 and the top border of the plate would be strictly smaller than the minimum waste. The optimal solution can be obtained by placing J_3 right on top of the plate. In the proof of Proposition III.2.1.2, items J_1 and J_2 are exchanged with items J_3. However, because of the precedence constraint in the first case and the defect in the second, this is not possible and the presented solutions are the only optimal solutions for each instance.

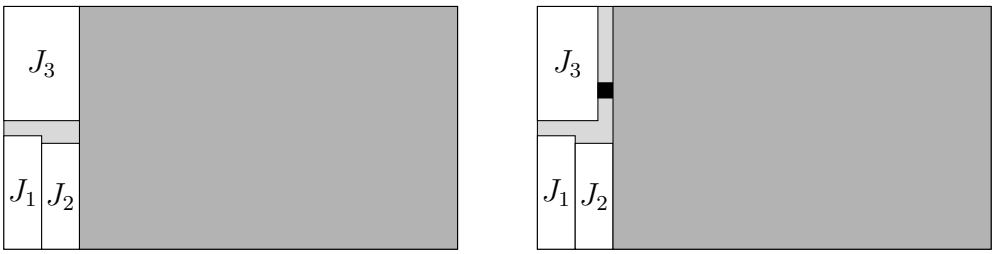


Figure III.9 – Counter-examples for the dominant set

III.2.2 General scheme

In the previous section, we showed that in most situations, packing items "in a corner" is dominant. Besides, more general dominance properties we could think of all increase the number of children of a partial solution beyond what is practically tractable. As we are looking for efficient heuristics, we assumed that this "in the corner"'-rule is a good compromise on which our tree generation can be based. Thus, we insert items in the order they are produced, ensuring that the precedence constraint is always satisfied.
The root node is the empty solution. Then we define the children of a node. Due to the 4-cut constraint, if we look at a solution rectangle defined by (1) the bottom of the plate or a 2-cut and (2) the next 2-cut (or the top of the plate if there is none) and (3) the left of the plate or a 1-cut or a 3-cut and (4) the next 3-cut (or 1-cut if there is none, or the right of the plate if there is no 1-cut), then this rectangle may contain either only waste, or an item without waste, or an item with waste above, or waste with an item above, or two items without waste. Therefore, our tree generation consists in packing the next such rectangle of the solution. This implies four types of insertions illustrated on Figure III.10 (current and previous cuts have been defined at the beginning of Section III.2; remember Figure III.4, page 98):

- inserting an item such that its bottom is the new previous 2-cut. These are the most common insertions. If there is one, the 4-cut will be the top of the item. This is how items 231, 334, 335, 232, *etc.* have been inserted.

- inserting an item such that its top is the new current 2-cut. This happens when the first type of insertion is not possible because the item would contain a defect. In this case, there will necessarily be a 4-cut for the bottom of the item. This is how items 6 and 56 have been inserted.

- inserting two items one above the other. Then the bottom of the first

item is the new previous 2-cut and the top of the second item is the new current 2-cut. There will be a 4-cut between the two items. Since no 5-cut is allowed, the two items must necessarily have the same length. This is how items 409 and 410 have been inserted.

- inserting a defect. This corresponds to the insertions before item 232, items 409 and 410, and item 247. Note that the defect on the right of item 8 does not need to be inserted.

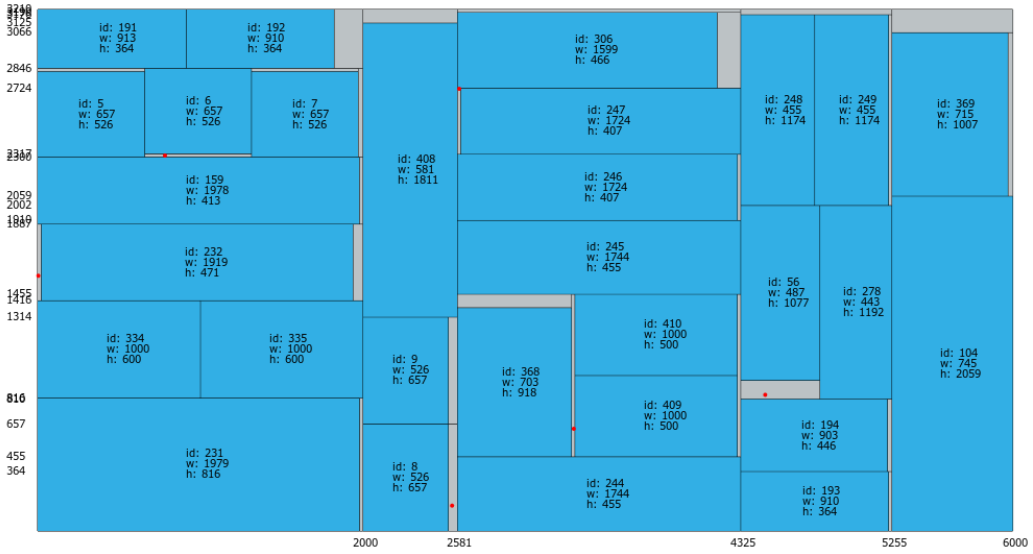


Figure III.10 – A plate of a solution

An insertion can be done at different depths:

- depth 0: the insertion is done on a new plate. This is how item 231 has been inserted.
- depth 1: the current 1-cut becomes the previous 1-cut and a new current 1-cut is created. This item is then at the bottom of the plate. This is how items 8, 244, 193 and 104 have been inserted.
- depth 2: the current 2-cut becomes the previous 2-cut and a new current 2-cut is created. This is how items 334, 159, 5, 191, 9, *etc.* have been inserted.
- depth 3: the current 3-cut becomes the previous 3-cut and a new current 3-cut is created. This is how items 335, 6, 7, 192, *etc.* have been inserted.

Finally, each item may be rotated.

III.2.3 Computing new current cut positions

When doing an insertion, the positions of the new current 1-cut, 2-cut and 3-cut need to be computed. The other cuts will not change. Here, we give some insights on how it is implemented. For an exhaustive explanation, please refer directly to the code[4].

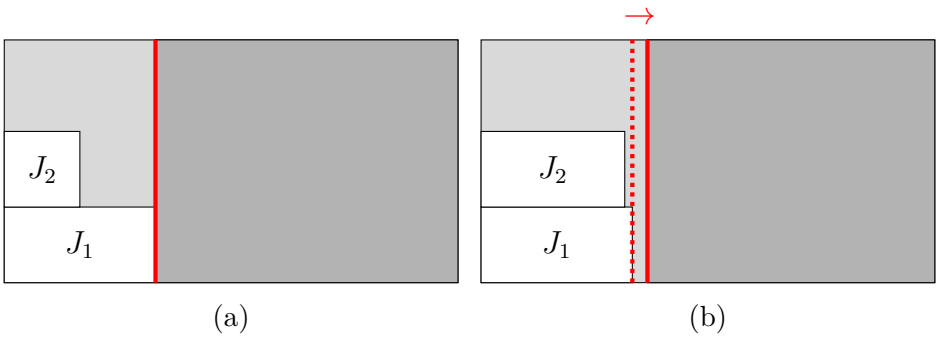


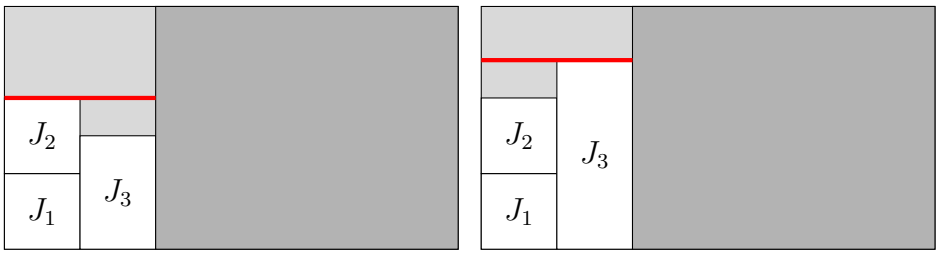
Figure III.11 – Computing the new current 1-cut

Figure III.11 illustrates two examples of the computation of the new current 1-cut after an insertion. In both cases, item J_1 is first inserted and then J_2. In the first case, the length of item J_2 is small enough so that the current 1-cut remains valid. In the second case, the length of item J_2 differs by less than the minimum waste from the length of item J_1: if the current 1-cut stayed the same, then the minimum waste constraint would be violated. Therefore, the current 1-cut needs to be moved to the right so that neither J_2 nor J_1 violate the minimum waste constraint; that is: its new position is $x_2(J_1) + w_{\min}$.
This example illustrates the need for the following definitions. Let $z_1(S)$ be such that

- $z_1(S) = 0$ iff to move the current 1-cut to the right, it should be moved by at least the minimum waste in order to respect the minimum waste constraint. This is the case in Figure III.11 after inserting item J_1. More generally, this happens when the current 1-cut is the right cut of an item.

- $z_1(S) = 1$ iff the current 1-cut can be moved to the right by any value without violating the minimum waste constraint. This is the case in solution (b) of Figure III.11 after inserting J_2. This may also happen because of defect insertions.

[4] https://github.com/fontanf/roadef2018



(a) feasible (b) infeasible

Figure III.12 – Illustration of the case $z_2(S) = 2$

Similarly, we define $z_2(S)$ for the current 2-cut; it can however take three different values. Figure III.12 illustrates two solutions where the first insertion is a 2 items insertion. Because of the 4-cut constraint, the top of item J_2 must necessarily be a 2-cut and cannot be a 4-cut. Therefore, in solution (a), the insertion of J_3 at depth 3 leads to a feasible solution, but in the second solution, the insertion of J_3 at depth 3 leads to an infeasible solution. In other words: once two items have been inserted, the position of the current 2-cut is fixed. Therefore, let $z_2(S)$ be such that

- $z_2(S) = 2$ iff the current 2-cut cannot be moved without violating the 4-cut constraint.
- otherwise:
  - $z_2(S) = 0$ iff to move the current 2-cut to the top, it should be moved by at least the minimum waste in order to respect the minimum waste constraint.
  - $z_2(S) = 1$ iff the current 2-cut can be move to the top by any value without violating the minimum waste constraint.

The value of $z_1(S)$ and $z_2(S)$ must be taken into account when computing the position of the new current cuts in order to insure generating feasible partial solutions.
When updating the position of the new current 2-cut, items inserted above a defect must also be taken into account. Indeed, if the position of the cut increases, then those items will also be moved. This is illustrated by Figure III.13. Initially, in (a), item J_1 is inserted above the first defect. When inserting item J_2, if the current 2-cut were moved to $y_2(J_2)$, then item J_1 would be moved on a defect, generating an infeasible solution. Therefore, the current 2-cut must be moved higher.

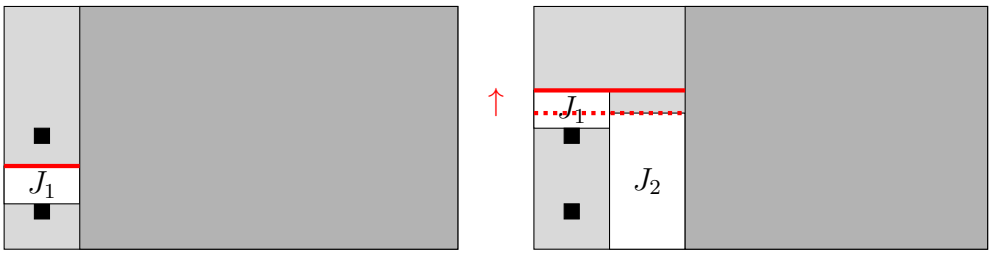


Figure III.13 – Updating the current 2-cut when items have been inserted above a defect

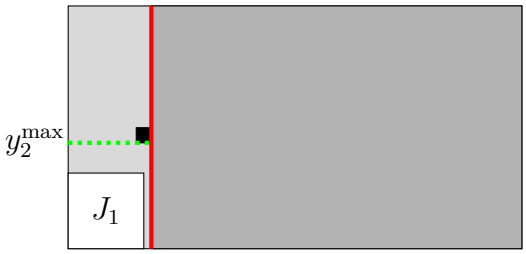


Figure III.14 – Updating the current 1-cut or 3-cut to avoid cutting through a defect

In the solution represented on Figure III.14, if the current 1-cut is placed at $x_2(J_1)$, then it intersects a defect. Therefore, it must be shifted to the right. Note that it must be shifted enough so that the minimum waste constraint is satisfied between the 3-cut at $x_2(J_1)$ and the current 1-cut. Furthermore, the 3-cut at $x_2(J_1)$ may also later intersect the defect if the current 2-cut is moved too high. Therefore, we also introduce and compute $y_2^{\max}(S)$ the maximum height of the current 2-cut in S.

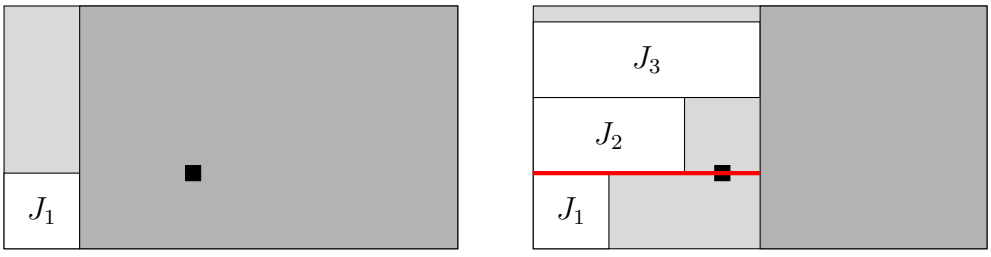


Figure III.15 – Avoiding cutting through defect with 2-cuts

Avoiding intersections between 2-cuts and defects is handled a bit differently. In the first solution of Figure III.15, the current 2-cut does not intersect the

defect yet. However, depending on the following insertions, that may happen later, as illustrated by the second solution. Therefore, we introduce $x_1^{\max}$ as the maximum position of the current 1-cut. Note that $x_1^{\max}$ cannot be computed right after inserting J_1 since the position of the current 2-cut is not fixed yet: an item higher than J_1 could be inserted at depth 3 and the 2-cut would be shifted higher and would not intersect the defect. Therefore, $x_1^{\max}$ is computed when the current 2-cut gets fixed. In the example, this corresponds to the insertion of J_2.

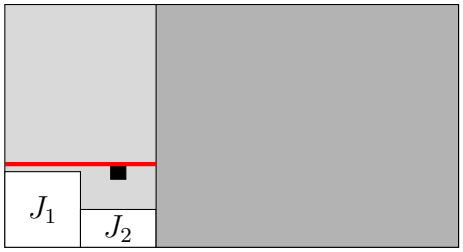


Figure III.16 – Avoiding cutting through defect with 2-cuts

If the current 2-cut intersects a defect, as illustrated on Figure III.16, then it is shifted to the top similarly to what happens when an item inserted above a defect intersects a defect.
Note that $x_1^{\max}$ is also used to handle the maximum 1-cut size constraint.
The branching scheme described in this section generates a high number a children for each node. Hence the need to cut dominated nodes or break symmetries.

III.2.4 Depth based cuts

Some rules illustrated on Figure III.17 are applied to avoid generating unpromising nodes:

- if an item can be inserted on the current plate, then the insertion in a new plate, *i.e.* with depth 0 is not considered;
- if an item can be inserted at depth 2 or 3 without increasing the current 1-cut, then the insertion with depth 1 is not considered;
- if an item can be inserted at depth 3 without increasing neither the current 1-cut nor the current 2-cut, then the insertions at lower depths are not considered.

Some other rules make it possible to avoid generating identical solutions:

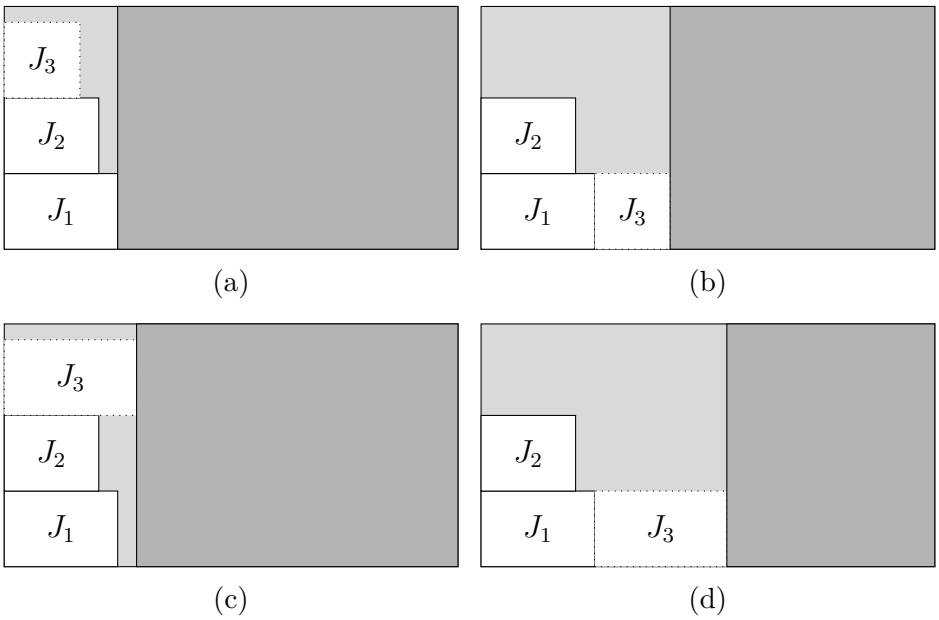


Figure III.17 – In solution (a), item J_3 can be inserted at depth 2 without increasing the current 1-cut. Therefore, solution (b) where it is inserted at depth 1 is not considered. In solution (c), J_3 can be inserted at depth 2, however, this increases the current 1-cut. Therefore, solution (d) where it is inserted at depth 1 is considered anyway.

- if the last insertion is a defect at depth d, then the next insertion must also happen at depth d;
- if the last insertion is a 2-item insertion at depth $d \neq 3$, then the next insertion must be at depth 3 (unless some tricky conditions that will not be described here are met).

III.2.5 Pseudo-dominance rule

When looking at a partial solution, we see a "front" defined by the previous 1-cut, the current 2-cut, the current 3-cut, the previous 2-cut and the current 1-cut (see Figure III.18).
The idea is that if the front of a partial solution S_1 is "before" the front of another partial solution S_2 that contains the same items, then we assume that S_1 dominates S_2 (see Figure III.19). More precisely, we use Algorithm 21 (in appendix at the end of this chapter, page 121) to determine if a solution dominates another one. This dominance is not exact (see example of Figure III.20) but it seems to be a good compromise. It is used in the main

algorithm to compare children of a node, and for all generated nodes in the algorithm dedicated to instances with only one or two chains.

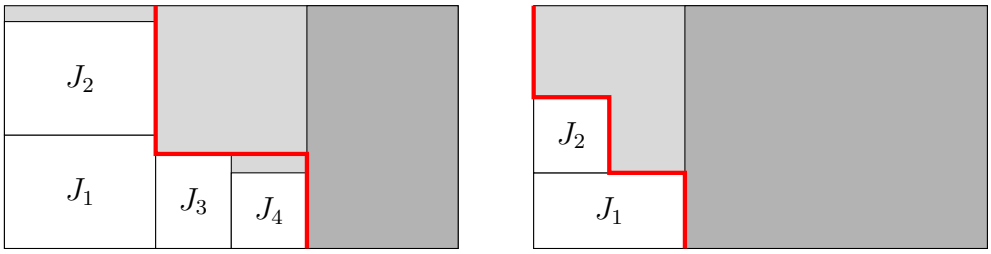


Figure III.18 – Illustration of the front of two partial solutions

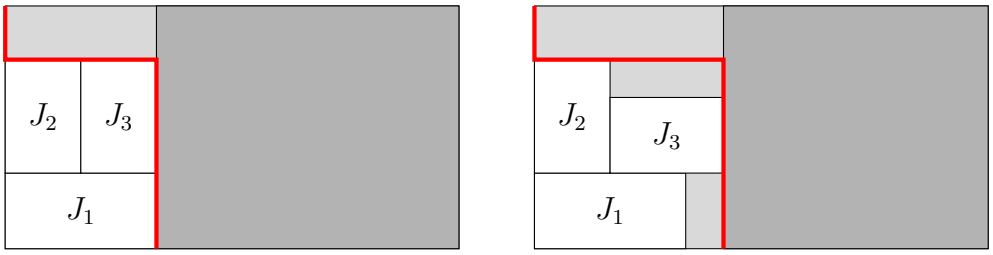


Figure III.19 – The first partial solution dominates the second one.

III.2.6 Breaking symmetries

Most instances of the challenge (i.e. 27 over 30) have more than 5 chains. Thus, they have too many sub-problems for a dynamic programming strategy. Therefore, we only use the pseudo-dominance rule between the children of a node, and not between nodes with different fathers. To compensate, we use some rules to avoid exploring equivalent nodes several times instead. A node is cut if a "k-block" (the items between two consecutive k-cuts) can be exchanged with the previous "k-block", *i.e.* without violating neither the defect constraint nor the precedence constraint, and if the minimum index of its items is lesser. Since the algorithm is constructive, we only need to check the penultimate and the antepenultimate "k-blocks". The current ones cannot be compared since they may still change later. Symmetry breaking is illustrated on Figures III.21 and III.22.

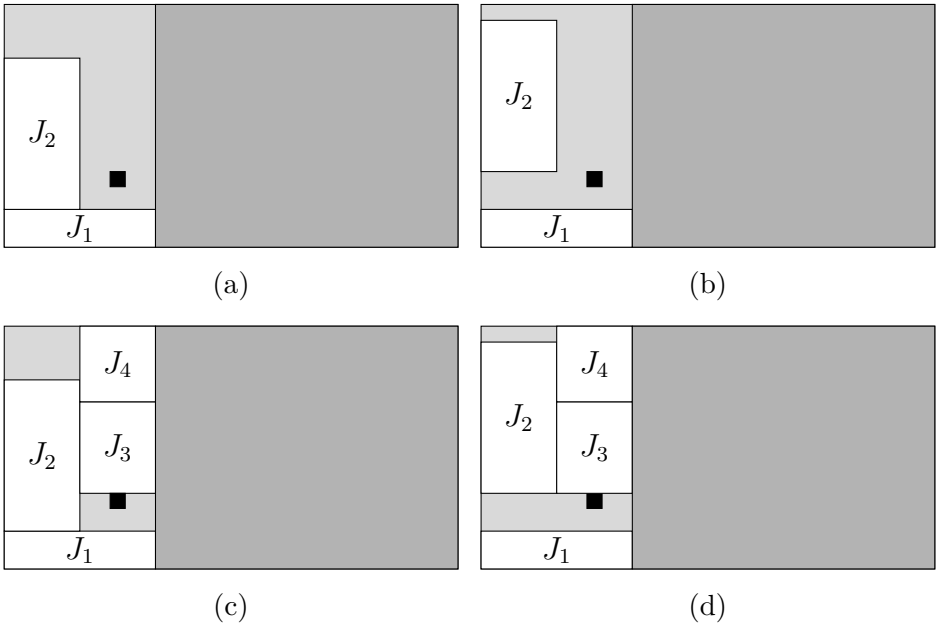


Figure III.20 – Consider an instance with 4 items and 1 chain. In solution (b), the defect has been inserted before J_2. The pseudo-dominance rule states that solution (a) dominates the second one. However, the solution (a) cannot lead to the optimal solution, since solution (c) violates the 4-cut constraint, whereas solution (b) can lead to solution (d) which is optimal.

III.3 Tree search

It is common in Operations Research to encode the search space as a tree. For instance Branch and Bound/Cut fully explores a tree where each node is a partial solution. Branch and Bound algorithms are complete tree searches designed to provide a guaranteed optimal value. But this is usually done at the price of storing a lot of information and not focusing on finding good solutions fast. On the other hand, greedy algorithms can be seen as a special kind of tree-search. They only look at the best next node until reaching a leaf, providing reasonable solutions very fast but without any guarantees.

Both methods have their quality and drawbacks. Once again, we opted for a compromise. Some tree searches, at the beginning, behave like greedy algorithms, then, as time goes, behave more and more like a Branch and Bound. Methods with this property are called in AI communities *anytime algorithms* because they can be stopped at any time and still provide good solutions. Some anytime algorithms are known in Operations Research. Indeed, Beam Search, an anytime method from AI, is popular in scheduling and packing

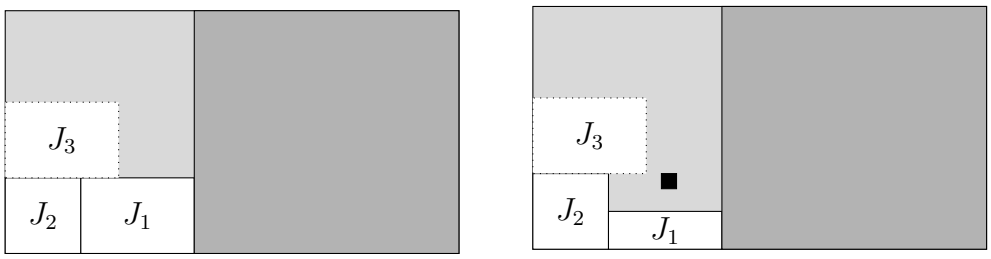


Figure III.21 – Illustration of symmetry breaking. In the first solution, items J_1 and J_2 can be exchanged. Therefore, the corresponding node is pruned. However, in the second solution, the two items cannot be exchanged because of the defect. Therefore, the solution is kept.

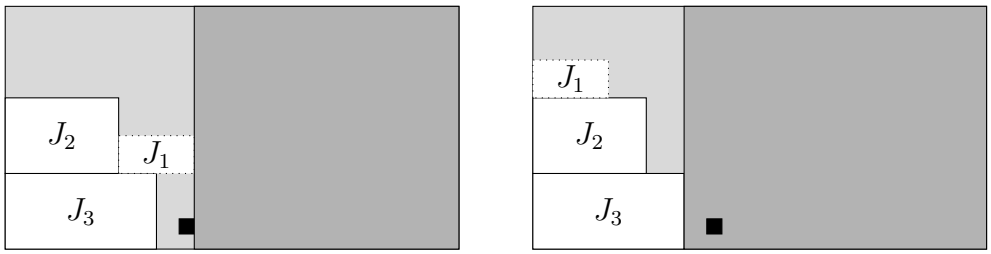


Figure III.22 – Illustration of symmetry breaking. In the first solution, before inserting item J_1, items J_2 and J_3 can be exchanged. However, the solution is not cut since the 2-block is not finished yet as item J_1 will be inserted after. However, item J_1 cannot be inserted at depth 2 as in the second solution.

communities. Also, we can note that ant colony optimization is a form of anytime tree search (Blum 2005). However, many more anytime tree searchs exist in AI. To the best of our knowledge, many of them are seldom used in Operations Research. We believed that importing anytime tree searches into Operations Research can lead to competitive methods. We did so, and the resulting method was ranked first in the final phase of the challenge.
In the previous section, we have presented how we generate a search tree encoding the glass cutting challenge. In this section, we present several classical anytime tree search algorithms from the AI community. Then we present a new anytime algorithm we used during the challenge, which is inspired by the AI tree search.

III.3.1 Anytime tree searches

Anytime tree searches are usually based on 3 classical searches: *Breadth First Search (BFS)*, *Depth First Search (DFS)* and *A**. For more information

about BFS and DFS, we refer the reader to Russell and Norvig 2016.
We quickly present A*. A* is guided by a lower bound (for a minimization problem) of a node $f(n) = g(n)+h(n)$ where $g(n)$ is the prefix cost and $h(n)$ an optimist estimate cost. A* considers a list of open nodes called *fringe*. At the beginning, the fringe contains only the root. At each iteration of the algorithm, A* extracts the node with the smallest lower bound, removes it from the fringe, and adds all its children into the fringe. It continues until it finds a feasible solution n (which is also optimal since the value of n is better than any other remaining node lower bound) or the fringe is empty (in this case, there is no feasible solution in the search tree). A* may take a lot of time and memory, however, the first solution it provides is optimal. It is summed-up in Algorithm 19 (see Russell and Norvig 2016 for more details).

Algorithm 19 A* algorithm

fringe $\leftarrow$ {root}
while fringe $\neq \emptyset$ **do**
    $n \leftarrow extractSmallestLowerBound$(fringe)
    if n is feasible **then return** n
    fringe $\leftarrow$ fringe $\setminus \{n\}$
    for all $v \in children(n)$ **do**
        fringe $\leftarrow$ fringe $\cup \{v\}$

In this section, we give a brief overview of the most popular anytime tree search algorithms that, we believe, can obtain good results in Operations Research problems. We implemented and compared those algorithms on the challenge instances. Most anytime tree searches (including the methods presented below) are based on one of the 3 classical searches. They modify the base search in order to make it anytime.

Beam Search (BS) was originally proposed by Rubin and Reddy 1977, first as *LOCUS Search*. Indeed, instead of considering all the nodes at a given depth, it only keeps the D best nodes within a given level. D is usually called the *beam size*. Beam Search is a generalization of both the greedy algorithm (when $D = 1$) and BFS (when $D = \infty$). Several variants of the Beam Search start with a small beam and run again the algorithm with a bigger beam size until the time limit. This process allows to provide good solutions fast and improve them in the later stages of the search. We also note that Beam Search is probably the most popular anytime tree search algorithm in Operations Research. It is present in several popular methods (see for instance Blum 2005, Ow and Morton 1988).

Simplified Memory-bounded A (SMA*)* was originally proposed by Russell 1992. To the best of our knowledge, SMA* is not popular in Operations Research. It consists in exploring the search tree as A*, opening the node n with the smallest lower bound. It adds to the fringe the child of n which has the smallest lower bound (and which is then removed from the list of children of n), and updates the lower bound of n and its ancestors if needed. n is discarded if it does not have any children left, otherwise, it is kept into the fringe with an updated lower bound. Like Beam Search, it considers a maximum fringe size D. If the fringe contains more than D nodes, SMA* discards the nodes which have the biggest lower bound. As Beam Search, SMA* generalizes the greedy algorithm if $D = 1$, and, SMA* generalizes A* if $D = \infty$.

III.3.2 The proposed anytime tree search: Memory Bounded A* (MBA*)

A* is known to minimize the cost estimate on nodes it opens. However it suffers from a large memory requirement since it has to store a large amount of nodes in the fringe. We propose a simple but yet powerful heuristic variant of A* that cuts less promising nodes if the fringe has more than D nodes. We call this new tree search algorithm *Memory Bounded A* (MBA*)*. Like Beam Search, it takes a parameter D. Like SMA*, if $D = 1$ it corresponds to a greedy algorithm and corresponds to A* if $D = \infty$. *MBA** outperformed the other procedures presented before on the challenge instances given the branching scheme described in Section III.2. The main difference between *MBA** and *SMA** is that *MBA** never opens partially a node. Indeed, SMA* stores partially explored nodes which makes the algorithm more difficult to implement. MBA* adds all the children of the selected node at once and then discards it.

Algorithm 20 Memory Bounded A* (MBA*)

```
1: fringe ← {root}
2: while fringe ≠ ∅ ∧ time < time limit do
3:     n ← extractBest(fringe)
4:     fringe ← fringe \ {n}
5:     for all v ∈ children(n) do
6:         fringe ← fringe ∪ {v}
7:     while |fringe| > D do
8:         n ← extractWorst(fringe)
9:         fringe ← fringe \ {n}
```

III.3.3 Guide functions

Tree search methods are dependent on well crafted guide functions. In the literature on tree search, many use lower bounds as guides since the first found solution is known to be optimal. However, for this problem, the search space is too big to prove optimality and using a lower bound may not be the best choice to find good solutions.
Indeed, during the development, we started by using the waste area, which is a lower bound, as a guide. We noticed that the resulting solutions packed small items on the first plates and big items on the last plates. These solutions had small waste area in the beginning but a lot in the last plates which made the solutions globally bad.
Another issue with using lower bounds is that it may not be relevant to compare two nodes from different depths of the tree. For example, a solution with only few items will always be considered more interesting than a solution containing almost all items.
We designed some "heuristic" guides to overcome these issues and obtained better solutions. We studied and use the two following guide functions:

- waste percentage: nodes containing big items become more interesting if they do not generate too much waste with respect of their own size;
- waste percentage/average item area packed: nodes containing big items are even more appealing.

We also tried several combinations of the above guides. The results were not promising and we only used those two on separate processors.

III.4 Complete algorithm

In the previous sections, we described the branching scheme and tree search algorithms. We now explain how we assemble these components together. Note that we take advantage of the multi-core architecture of the computer used for evaluating algorithms in the challenge.
First, we distinguish the case where the instance has two chains or less. In this case, we run the A* algorithm with dynamic programming based on the pseudo-dominance rule described in Section III.2.5 (we call it DPA*). If an instance is only composed of one chain, and the instance contains less than 700 items (this information was given on the challenge description), then the algorithm solves it optimally according to the branching scheme and the pseudo-dominance rule in less than a second. If the instance is composed of

two chains, there is at most $350^2 = 122,500$ sub-problems and DPA* solves it optimally in a few seconds.
If there is strictly more than two chains, we do not use dynamic programming since it would cost too much in memory. Instead, we run 4 threads in parallel, each one running a restarting MBA* with a given growth factor and guide function. Restarting means that, initially, the maximum size of the fringe is small (3 for example), and each time MBA* terminates, it is restarted with a greater maximal size (multiplied by the growth factor). If the growth factor is 2, the maximal size doubles at each iteration. We use growth factors 1.33 and 1.5 and the two guide functions described in Section III.3.3.
All the threads share the information of the best solution found. If one finds a better solution, the others can exploit this information to perform more cuts and globally perform better together than alone.

III.5 Computational experiments

We now present computational experiments of our algorithm on the challenge instances. They have been performed on a computer with an Intel Core i7-4790 CPU @ 3.60 GHz × 8 processor with 32 Go of RAM. This configuration is similar to the one of the challenge.
The instances are separated into three datasets. Dataset A was published during the qualification phase. It contains three trivial instances, some instances of moderate size containing between 30 and 130 items and three larger ones containing about 300 items. Dataset B was published after the qualification phase and was used for the evaluation during the final phase, along with dataset X which was unknown until the results were announced. They both include larger instances containing on average 300 items. Most instances contain between 10 and 15 chains. Three instances have exactly two chains and five instances have no precedence constaints.
Since the challenge, a few adjustments have been made (the version presented in this chapter is the current one). Therefore, the results presented here slightly differ from the results obtained during the final phase. Compared to the challenge version, the current version performs better: the total waste on datasets B and X decreases from 493 600 549 for the challenge version to 469 910 749 for the current one.
Figures III.23, III.24 and III.25 sum up computational experiments. Columns "Final phase best 180s" and "Final phase best 3600s" contain the values of the best solutions found by any team during the final phase (note that dataset A was not used in the final phase). A result annotated with a star indicates that this solution was found by our algorithm during the final phase of the

challenge. Column "MBA*/DPA* 3600" contains the solution values found by the current version of our algorithm in 3600 seconds. Finally, column "Best known" contains the values of the best solutions up to our knowledge. They may have been found during the development of the algorithm, when the execution time is longer than 3600s or by other teams.
In 180 seconds, the algorithm we submitted was ranked first on 17 and second on 10 out of the 30 instances used for the evaluation. In 3600 seconds, it was ranked first on 20 and second on 7. The current version finds the best known solutions on 14 instances of dataset A in less than 3600 seconds. It also finds the best known solutions on instances with exactly two chains in less than 180 seconds, showing the effectiveness of DPA*. Finally, even if it is not indicated in the table, on most of the instances, if the algorithm is run longer, for example 2 hours, the solutions will still be improved.

III.6 Conclusions and perspectives

We have proposed in this chapter an anytime tree search algorithm called MBA*. It performs successive iterations and restarts when the search tree is emptied. During the first iterations, it performs aggressive cuts and behaves like a greedy algorithm. As iterations go, the algorithm performs less heuristic cuts, enabling it to find better solutions. Possibly, the algorithm will perform an iteration with no heuristic cut, finding the best solution available in the branching scheme.
We also provide a framework to design anytime tree search algorithms. The algorithm design can be split into two parts:

Search tree design: Implicit definition of the search tree. It provides the root, how to generate children of a given node, cuts, lower bounds and guides.

Search tree exploration: A generic anytime tree search that explores the search tree previously defined. Those tree searches are well known in AI and we used them in the challenge.

This framework allowed us to design separate parts, consequently, helping us to do quick prototyping. We implemented 5 different branching schemes to encode the search tree until we found a good compromise in terms of speed and quality. We also implemented many search algorithms (Beam Search, Beam Backtrack, A*, MBA*, SMA*) and kept the one giving the best results. We do not want to draw premature conclusion, but we tried to understand the success of the algorithm:

INST	Comments	Final phase best 180s	MBA*/DPA* 180s	Final phase best 3600s	MBA*/DPA* 3600s	Best known
A1	Trivial	-	425 486	-	425 486	425 486
A2	No prec	-	9 506 669	-	4 383 509	4 383 509
A3		-	2 651 880	-	2 651 880	2 651 880
A4		-	3 024 240	-	2 924 730	2 924 730
A5		-	2 924 730	-	3 283 653	3 017 223
A6		-	3 389 640	-	3 225 930	3 188 646
A7		-	4 703 760	-	4 334 610	3 920 520
A8		-	9 691 844	-	8 378 954	8 378 954
A9		-	2 664 276	-	2 664 276	2 664 276
A10		-	4 084 381	-	4 084 381	4 084 381
A11		-	4 660 669	-	4 622 149	4 358 929
A12		-	2 056 504	-	1 879 954	1 879 954
A13		-	10 226 883	-	9 440 433	9 331 293
A14		-	11 686 638	-	10 383 378	10 383 378
A15		-	12 918 611	-	11 108 171	10 828 901
A16	Trivial	-	3 380 333	-	3 380 333	3 380 333
A17	2 chains	-	3 617 251	-	3 617 251	3 617 251
A18		-	5 596 728	-	4 983 618	4 983 618
A19		-	3 654 374	-	3 323 744	3 323 744
A20	Trivial	-	1 467 925	-	1 467 925	1 467 925

Figure III.23 – Computational experiments - Dataset A

INST	Comments	Final phase best 180s	MBA*/DPA* 180s	Final phase best 3600s	MBA*/DPA* 3600s	Best known
B1	No prec	3 232 698	3 765 558	**2 661 318***	3 136 398	2 661 318
B2		15 635 435*	14 312 915	13 674 125*	13 398 065	11 931 095
B3		20 540 813	19 786 463	18 191 093	17 093 273	15 786 803
B4		8 269 045*	8 323 615	8 269 045*	7 973 725	7 315 675
B5	2 chains	72 155 615	72 155 615	**72 155 615**	**72 155 615**	72 155 615
B6		12 116 527*	12 488 887	11 195 257*	11 089 327	10 800 427
B7	No prec	9 601 299	9 177 579	8 355 819*	7 678 509	6 628 839
B8		17 865 559*	17 152 939	16 067 959	15 840 049	14 398 759
B9		18 502 147	19 969 117	17 484 577	17 474 947	16 495 897
B10		26 012 183	26 904 563	**21 951 533***	23 065 403	21 951 533
B11		25 251 890	27 312 710	22 584 380	23 820 230	20 626 280
B12		15 868 657*	13 734 007	13 958 707*	13 120 897	12 514 207
B13		28 349 055*	27 360 375	24 471 375*	23 078 235	22 657 725
B14		9 346 480*	9 442 780	8 656 330*	8 377 060	8 023 960
B15		27 794 441*	24 568 391	24 517 031*	23 088 581	22 619 921

Figure III.24 – Computational experiments - Dataset B

INST	Comments	Final phase best 180s	MBA*/DPA* 180s	Final phase best 3600s	MBA*/DPA* 3600s	Best known
X1		15 508 097*	15 302 657	14 127 797*	14 127 797	13 720 127
X2	No prec	6 034 937	6 083 087	5 434 667*	4 879 337	4 795 877
X3		8 285 206*	7 649 626	7 473 076*	7 180 966	6 837 496
X4		12 182 072	15 488 372	**11 405 252**	13 366 562	11 405 252
X5		5 081 297	4 988 207	4 712 147	4 715 357	4 522 757
X6		12 565 673	11 031 293	10 363 613*	9 496 913	9 365 303
X7		22 443 360*	22 876 710	21 127 260	21 191 460	20 568 720
X8	2 chains	24 788 661*	22 265 601	24 788 661*	**22 265 601**	22 265 601
X9		22 251 225*	22 312 215	20 167 935	20 479 305	20 039 535
X10		20 110 472*	18 778 322	17 824 952*	17 186 162	16 865 162
X11		13 489 692*	12 802 752	12 417 552*	11 676 042	11 011 572
X12		11 963 845*	12 358 675	10 583 545*	10 503 295	10 246 495
X13		15 950 172	14 345 172	13 533 042*	13 125 372	12 130 272
X14		8 889 542*	8 591 012	8 013 212*	7 644 062	7 422 572
X15		13 990 194	13 710 924	11 682 204	11 682 204	10 882 914

Figure III.25 – Computational experiments - Dataset X

- For this problem, it seems more difficult to design local moves that keep the locality property of solutions, compared to more classical Operations Research problems. Therefore, constructive algorithms seem more suited, or at least more intuitive to develop, than local search algorithms. This assumption should nevertheless be put in perspective, since the algorithm ranked second in the challenge is based on local search and achieves very close results.

- When branching, there does not seem to be eliminatory choices. If a move is good locally, it should not be very bad globally. Even if the choice is not optimal, the very high number of possible children makes it possible to find a solution with similar cost anyway. This is true under the condition that all those children are implemented, *i.e.* that the branching scheme allows to reach the maximum number of configurations (while breaking symmetries). As shown in Section III.2, we gave a lot of attention to this part.

- The problem has natural guide functions to compare solutions at different depths of the tree. Therefore, MBA* seems more suited than Beam Search.

- The two parts design makes the code easier to write and to debug. In the context of an industrial problem with many complex constraints to satifsy, this may be crucial.

The results of the challenge prove that anytime tree searches can be competitive with local search based methods. We believe that they can be suited for other Operations Research problems, and particularly industrial ones with more complex constraints. Indeed, those constraints can weaken the locality property of solutions whereas they might be easily handled by a tree search approach. It should be worth experimenting such methods, especially MBA*, on those problems.

References

Blum (2005). "Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling". In: *Computers & Operations Research* 32.6, pp. 1565–1591. DOI: 10.1016/j.cor.2003.11.018.

Ow and Morton (1988). "Filtered beam search in scheduling". In: *International Journal of Production Research* 26.1, pp. 35–62. DOI: 10.1080/00207548808947840.

Algorithm 21 Pseudo-dominance algorithm

procedure DOMINATES(S_1, S_2)
  if $n(S_1) < n(S_2)$ **then** ▷ If S_1 has strictly less plates,
    return true ▷ then it dominates S_2.
  if $i(S_1) > i(S_2)$ **then** ▷ If S_1 has strictly more plates,
    return false ▷ then it does not dominate S_2.
  $x \leftarrow x_1^{\text{curr}}(S_1)$ ▷ x_1^{curr} may slightly be corrected.
  if $z_1(S_1) = 0$ and $x \neq W$ **then**
    if $z_1(S_2) \neq 0$ or $x \neq x_1^{\text{curr}}(S_2)$ **then**
      $x \leftarrow x + w_{\min}$
  $y \leftarrow y_2^{\text{curr}}(S_1)$ ▷ y_2^{curr} may slightly be corrected.
  if $z_2(S_1) = 0$ and $y \neq H$ **then**
    if $z_2(S_2) \neq 0$ or $y \neq y_2^{\text{curr}}(S_2)$ **then**
      $y \leftarrow y + w_{\min}$
  ▷ Comparison of the fronts.
  if $y_2^{\text{curr}}(S_2) \neq H$ and $x_1^{\text{prev}}(S_1) > x_1^{\text{prev}}(S_2)$ **then**
    return false
  if $x > x_1^{\text{curr}}(S_2)$ **then**
    return false
  if $y_2^{\text{prev}}(S_2) < y_2^{\text{prev}} S_1$ **then**
    if $x > x_3^{\text{curr}}(S_2)$ **then**
      return false
  else if $y_2^{\text{prev}}(S_2) < y$ **then**
    if $x_3^{\text{curr}}(S_1) > x_3^{\text{curr}}(S_2)$ **then**
      return false
  else
    if $x_1^{\text{prev}}(S_1) > x_3^{\text{curr}}(S_2)$ **then**
      return false
  if $y_2^{\text{curr}}(S_2) < y_2^{\text{prev}} S_1$ **then**
    if $x > x_1^{\text{prev}}(S_2)$ **then**
      return false
  else if $y_2^{\text{curr}}(S_2) < y$ **then**
    if $x_3^{\text{curr}}(S_1) > x_1^{\text{prev}}(S_2)$ **then**
      return false

Rubin and Reddy (1977). "The LOCUS Model of Search and its Use in Image Interpretation." In: *IJCAI*. Vol. 2, pp. 590–595.

Russell (1992). "E cient memory-bounded search methods". In: *ECAI-1992, Vienna, Austria*.

Russell and Norvig (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,

Chapter IV

Conclusion

In Chapter I, we studied a category of scheduling problems which are of theoretical and practical interest: processing time dependent profit maximization scheduling problems. We discussed some unusual properties and we proposed several piecewise-linear models as references. Then we underlined the complexity hierarchy that exists among those models. We proved NP-completeness of most general cases based on reductions to the Knapsack Problem and to $Pm \mid\mid C_{\max}$. We also proposed Dynamic Programming algorithms, list algorithms and b-matching formulation based algorithms running in polynomial time for many special cases.
Still, several other cases remain open; we give here some noticeable examples. $P \mid \text{LPST} \mid -\sum w_j(p_j)$ and $P \mid \text{LPST}, p_j^{\min}{=}p^{\min} \mid -\sum w_j(p_j)$ seem really simple and yet, we could not find polynomial algorithms that solve them. $1 \mid \text{LBPST}, p^{\min}{=}p^{\min} \mid -\sum w_j(p_j)$ corresponds to a fractional knapsack with a fixed setup cost. It seems related to the Cardinality Constrained Knapsack Problem which Farias Jr and Nemhauser 2003 proved to be NP-complete. $P \mid \text{LBP}, p_j^{\max}{=}p^{\max} \mid -\sum w_j(p_j)$ lies between $P \mid p_j{=}p \mid \sum w_jU_j$ and $P \mid p_j{=}p, \text{pmtn} \mid \sum w_jU_j$ that were proved to be respectively solvable in polynomial time and NP-complete (Brucker and Kravchenko 1999). Finally, $1 \mid \text{LP}, r_j \mid -\sum w_j(p_j)$ is also a very simple and yet open problem.
In processing time dependent profit maximization scheduling problems, the non-linearities in the profit functions avoid linear programming formulations, and the controllable processing times make it difficult to design reductions to NP-complete problems. Also, several cases have been proven to be polynomial when the number of parallel machines is fixed (Pm) or when all deadlines are equal ($d_j{=}d$), but the algorithms do not generalize to the cases with an unbounded number of parallel machines (P) or when deadlines are distinct.

In Chapter II, we studied a practical star observation scheduling problem. We continued the work started by Catusse et al. 2016 by integrating all practical constraints and designing and implementing algorithms more suited

for industrial contexts (simpler, anytime and more flexible).
We proposed a simple and general Large Neighborhood Search (LNS) scheme for star observation scheduling problems. The algorithm reduced the resolution of a specific variant to the resolution of a Single Night Timing Problem (SNTP) for which algorithms are much easier to develop and adapt. We illustrated this by implementing several algorithms solving SNTP for different variants of star observation scheduling problems.
Our Large Neighborhood Search algorithm is relevant because it clearly outperforms both the previous Local Search algorithm on literature instances for the pure problem, and the currently used Simulated Annealing algorithms on real instances. Besides, it is simpler and more easily adaptable than both. Compared to the Branch-and-Price algorithm, it provides solutions within 2% of optimality in less than 2 seconds, while the Branch-and-Price will need about 10 minutes to find its first solution on real sized instances. However, even if the Large Neighborhood Search continues improving the solution over time, the solution found by the Branch-and-Price is almost always the optimal one and the Large Neighborhood Search cannot find it within the same amount of time. Further research could for example focus on diversification techniques to improve this aspect.
Theoretically, allowing flexible observation durations necessarily improves the value of the optimal solution. In practice, on the given instances, LNS indeed finds better solutions in theses cases. However, the additional gain is rather moderate.
The algorithm is now implemented in the software used by the astrophysicists, and we hope it will be used in production very soon.

In Chapter III, we described the algorithm we submitted for the final phase of the ROADEF/EURO challenge 2018. This edition of the challenge was dedicated to a cutting optimization problem in collaboration with Saint-Gobain
We proposed an anytime tree search algorithm called MBA*. It performs successive iterations and restarts when the search tree is emptied. During the first iterations, it performs aggressive cuts and behaves like a greedy algorithm. As iterations go, the algorithm performs less heuristic cuts, enabling it to find better solutions. Possibly, the algorithm will perform an iteration with no heuristic cut, finding the best solution available in the branching scheme.
The results of the challenge prove that anytime tree searches can be competitive with local search based methods. We believe that they can be suited for other Operations Research problems, and particularly industrial ones with more complex constraints. Indeed, those constraints can weaken the locality

property of solutions whereas they might be easily handled by a tree search approach. It should be worth experimenting such methods, especially MBA*, on those problems.
The problem of the challenge is also interesting in itself. It is close to two-dimensional packing problems from the literature, but it adds several constraints that makes it harder to solve. The gap between the solutions returned by our algorithm in 1 hour and the best known solutions (not the optimal ones!) is more than 7%, which is high for an optimization problem.
The objective is also interesting. Traditionally, in bin packing problems, the objective is only to minimize the number of bins. Thus, unless the instance is intentionnaly built to avoid it, the lower bound obtained by column generation usually matches the optimal value and large problems are solved exactly. The bin packing problem is one of the NP-complete problem with the smallest input size and therefore can be deeply and almost exhaustively studied. The bin packing problems with leftovers offers a problem with the same input, but harder to solve. Some variants of this problem have been studied (see Cherri et al. 2014 for a review).

As stated in the introduction of the manuscript, the three chapters can be read independently. However, looking at them together gives some interesting perspectives.
In Chapter I we studied the complexity of a sub-problem of the real life application presented in Chapter II. The main conclusion of Chapter I can be stated as follows: "those problems are hard". Indeed, even when cases can be solved in polynomial time, the complexity is often deterrent and not suited for practical use. That is why, in Chapter II, we did not tackle this sub-problem directly, as it were before, but rather reduced the problem to one without the job selection property. And thus, it becomes much easier to solve.
It is also interesting to compare Chapter II and III as they show two ways of doing practical Operations Research. In both chapters, the goal was to design algorithms for a real world problem. However, in Chapter III, the problem and the instances are well defined and the objective is to develop the best possible algorithm for them, whereas in Chapter II, the problem is evolving, changing, and the objective is to design an efficient enough algorithm that can be easily adapted. But still, the algorithm developed in Chapter III remains rather simple and easily adaptable, and the one developed in Chapter II has been proven to be competitive with previous state of the art methods.

References

Brucker and Kravchenko (1999). *Preemption Can Make Parallel Machine Scheduling Problems Hard*.

Catusse, Cambazard, Brauner, Lemaire, Penz, Lagrange, and Rubini (2016). "A Branch-And-Price Algorithm for Scheduling Observations on a Telescope". In: *Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI Press, pp. 3060–3066.

Cherri, Arenales, Yanasse, Poldi, and Gonçalves Vianna (2014). "The one-dimensional cutting stock problem with usable leftovers – A survey". In: *European Journal of Operational Research* 236.2, pp. 395–402. DOI: 10.1016/j.ejor.2013.11.026.

Farias Jr and Nemhauser (2003). "A polyhedral study of the cardinality constrained knapsack problem". en. In: *Mathematical Programming* 96.3, pp. 439–467. DOI: 10.1007/s10107-003-0420-8.

Summary

Large telescopes are few and observing time is a precious resource subject to a strong pressure from a competitive community. Besides, a growing number of astronomical projects require surveying a large number of targets during discontinuous runs spread over few months or years. This thesis is the result of a collaboration between operations researchers from G-SCOP laboratory and astrophysicists from IPAG research unit. Its goal was to optimize the schedule of star observations on telescopes, in particular, on the Very Large Telescope. The first chapter contains a theoretical study of the complexity of several variants of processing time dependent profit maximization scheduling problems. Those problems appear *inter alia* as sub-problem of the real life star observation scheduling problem studied in the second chapter, for which we developed a fast and efficient Large Neighborhood Search algorithm. The third chapter is not related to the star observation scheduling problem. It describes the Tree Search based algorithm we developed for the glass cutting problem raised by the company Saint-Gobain for the ROADEF/EURO challenge 2018.

Résumé

Il existe peu de grands télescopes et le temps d'observation est une ressource précieuse sujette à une forte pression de la part d'une communauté concurrentielle. En outre, un nombre croissant de projets astronomiques nécessitent l'observation d'un grand nombre d'objets célestes pendant des runs discontinus répartis sur plusieurs mois ou années. Cette thèse est le fruit d'une collaboration entre des chercheurs en recherche opérationnelle du laboratoire G-SCOP et des astrophysiciens de l'unité de recherche IPAG. Son but était d'optimiser l'ordonnancement d'observations d'étoiles sur les télescopes, en particulier, sur le Très Grand Télescope. Le premier chapitre contient une étude théorique de la complexité de plusieurs variantes de problèmes d'ordonnancement de maximisation de profits dépendants du temps d'exécution. Ces problèmes apparaissent entre autres comme sous-problèmes du problème réel d'ordonnancement d'observations d'étoiles étudié dans le deuxième chapitre, pour lequel nous avons développé une recherche locale à voisinage large rapide et efficace. Le troisième chapitre ne concerne pas le problème d'ordonnancement d'observations d'étoiles. Il décrit l'algorithme de recherche arborescente que nous avons développé pour le problème de découpe de verre posé par l'entreprise Saint-Gobain pour le challenge ROADEF/EURO 2018.