Complexity of scheduling tasks with processing time dependent profit

Florian Fontan, Pierre Lemaire, Nadia Brauner

Complexity of scheduling tasks with processing time dependent profit

Florian Fontan, Pierre Lemaire, Nadia Brauner

Univ. Grenoble Alpes, CNRS, Grenoble INP[1]*, G-SCOP, 38000 Grenoble, France*

Abstract

In this paper, we introduce and study scheduling problems, where tasks yield a processing time dependent profit. In these problems, the processing time of a task is not given as input but must be decided; the profit from a task then depends on its allocated processing time and the objective is to maximize the total profit. These problems originate from astrophysics.

First, we state formally the problem and propose several profit functions, including non-continuous and non-concave ones. Next, we prove and discuss some generic properties on the structure of solutions. Then, we study the complexity of the main models, proving the NP-completeness of some, and proposing polynomial algorithms for others.

Keywords: complexity, scheduling, processing time dependent profit, polynomial algorithms, controllable processing times
2010 MSC: 00-01, 99-00

In this paper, 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 [1], but it could model various situations. Astrophysicists want to schedule observations on a telescope and, for each possible target (star), there exist a time-window 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 in [2], 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, and many others, can be modelled by processing time dependent profits: hence there is a need to study such models and, first of all, their complexity. That is the point of this article.

[1] Institute of Engineering Univ. Grenoble Alpes

In the remainder of this paper, we first state formally the problem and propose several profit functions (Section 1); then, we prove some generic properties (Section 2) moving on to NP-completeness results (Section 3) before considering polynomial cases (Section 4).

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 1 shows three examples of profit functions. Note that, for some settings, it may be impossible to schedule all jobs, even with null processing time (for example when there are release dates) and that this is not a regular criterion. Also, note that the instance size (as input of a Turing machine) depends on how w_j are defined. In the remainder of this paper, we call this problem and its variants Processing time Dependent Profit Scheduling Problems (PDPSP).

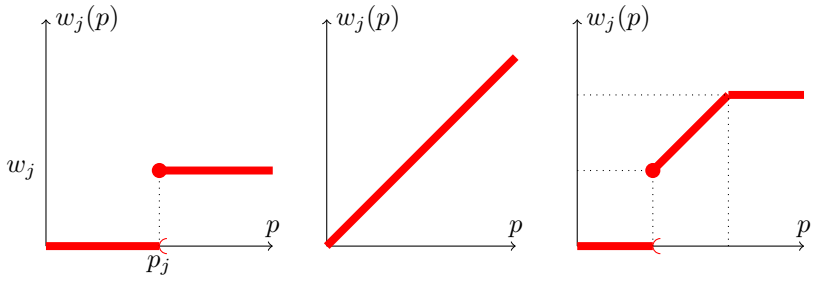


Figure 1: Examples of profit functions of: (a) a classical scheduling problem ($w_j(p) = w_j$ if $p \geq p_j$, 0 otherwise); (b) basic (linear profit) problem; (c) the star observation problem

Some cases of processing time dependent profit are found in the literature under the keywords "Increasing Reward with Increasing Service". In particular, [3] solves the case of non-decreasing concave models with deadlines (the idea is to allocate service to the task with the highest marginal reward till its

marginal reward equals that of the task with the second-highest marginal reward, and then to allocate service to both till both their marginal rewards equal that of the task with the third-highest marginal reward, and so on).

PDPSP are connected to problems where processing times are controllable by allocating a resource; those have received a lot of attention (see [4, 5] for surveys). They are particularly close to those problems where the objective is to minimize the amount of ressources that are used, but they fundamentally differ because, in the latter case, all jobs have to be scheduled (unlike PDPSP).

PDPSP are also related to scheduling problems with late work criteria, noted Y_w (see [6] for a survey), in which only the units executed after the due dates are penalized.

We propose now several profit functions that are studied thereafter; to this extent, we adapt the 3-field notation (see e.g. [7]) 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.

In this paper, we consider the profit functions presented in Figure 2 (the one corresponding to the original astrophysics problem is the last). 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.

All these models can be generalized into 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);

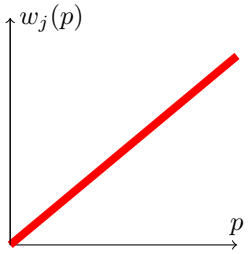
(3) $w_j^{k+1} \geq w_j^k + b_j^k(p_j^{k+1} - p_j^k)$ (profit is non-decreasing);

(4) all data are integers.

A PLP can be any non-decreasing piecewise-linear function starting at 0 and may be neither continuous nor concave; Figure 3 shows the different possible configurations of pieces. The assumption that input data are integers holds for every model (and is further discussed later on, see Section 2). Note that if p is an integer, $w_j(p)$ is also an integer for all profit functions defined previously; in particular, $w_j^{\max}$ is also an integer. Also, note that, for a given p, computing $w_j(p)$ requires $O(\log K_j)$ time to search for the piece k such that $p_j \in [p_j^k, p_j^{k+1}[$.

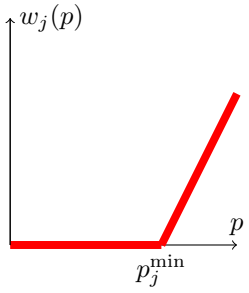
Linear Profit (LP)

$$w_j(p) = b_j p$$



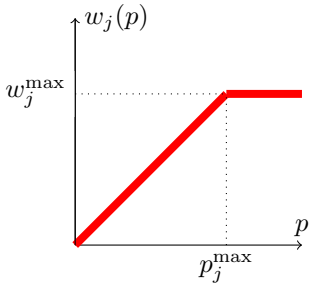
Linear Profit with Setup Time (LPST)

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



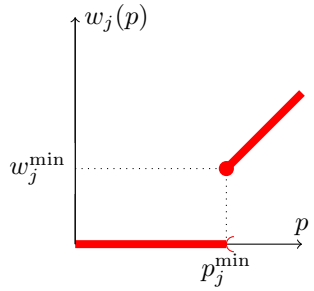
Linear Bounded Profit (LBP)

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



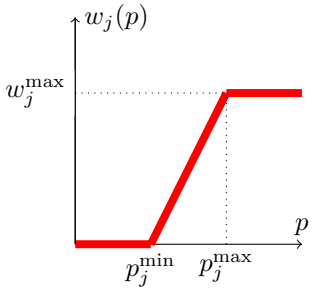
Linear Profit with Setup Time and Initial Profit (LPSTIP)

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



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)

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

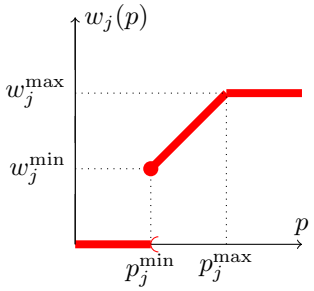


Figure 2: Profit functions

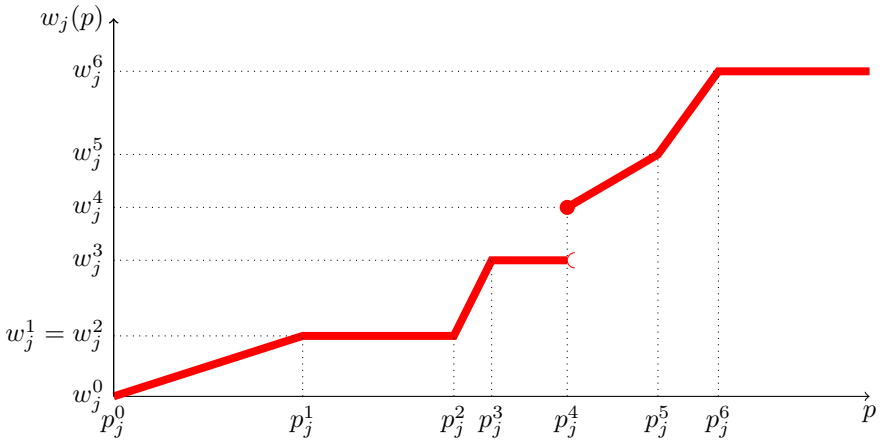


Figure 3: A PLP function showing the different types of pieces

In this paper, we study the complexity of the various PDPSP with a focus on the PLP model. In the process, we shall deduce the complexity of several of the particular cases presented above.

2. Notations and general results

Some notations are used consistently throughout the rest of this article. 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 paper, 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 the previous section strongly relate one to another, and there is indeed a whole complexity hierarchy that exists among them and is represented by Figure 4. This hierarchy is useful to exhibit maximally-polynomial or minimally-NP-complete cases.

Then, a generic dominance property exists:

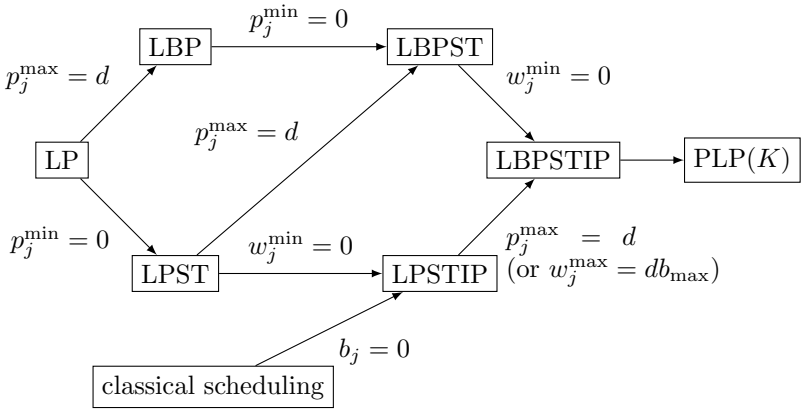


Figure 4: Complexity hierarchy of PDPSP problems. An arc from A to B means that A is a special case of B, and a label indicates the particular settings of B that corresponds to A. $PLP(K)$ is the particular case when the number of pieces is bounded by K.

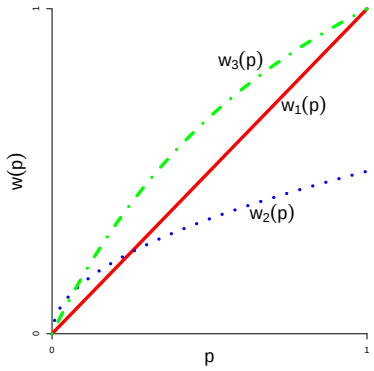
Lemma 2.1. *For $P\,|\,|\,-\sum w_j(p_j)$, the set of solutions for which all tasks are scheduled (possibly with $p_j = 0$) and such that, on each machine, tasks are scheduled in non-decreasing order of their deadlines and without idle time, is dominant.*

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, 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 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 (simple such cases are depicted in Figure 5). However, for the PLP model, as we assume integer



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: $p_1 = \sqrt{2} - 1$, $p_2 = 2 - \sqrt{2}$.

Figure 5: Two simple examples of non-linear PDPSP for which an optimal solution must have some tasks with non-integer start or end dates.

data and (piecewise) linear profit, solutions with integer processing times are indeed dominant:

Lemma 2.2. *For $P \mid PLP \mid -\sum w_j(p_j)$, 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 2.1.

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 two first 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 2.2 still does not provide a direct optimal solution, but we can develop a dynamic programming algorithm on it:

Theorem 2.3. *Algorithm 1 returns an optimal solution to* $P \mid PLP \mid -\sum w_j(p_j)$ *in time* $O(n(md^{m+1} + \log n))$.

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, $j = 1, \ldots, n$ and for all $p = 0, \ldots, d_j$
3: Let $\mathcal{C} = \{C \in \{0, \ldots, d\}^m, C_1 \leq \cdots \leq C_m\}$. Return $\max_{C \in \mathcal{C}} f^*(n, C)$ such that:

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

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 $e_k^i = 0$ for all $k \neq i$.

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 2.2, 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 $O(nmd^{m+1} + n \log n)$. □

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 2.2 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_j^k, and only a few will not. If two consecutive tasks are not scheduled during a p_j^k, 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_j^k. We define $\mathcal{P}_j$ the set of "interesting" processing times of task T_j as follows:

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

with $\rho_j^1 \leq \cdots \leq \rho_j^{K_j^\rho}$ 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 6; 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 2.4. *For $P \mid PLP \mid -\sum w_j(p_j)$, 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

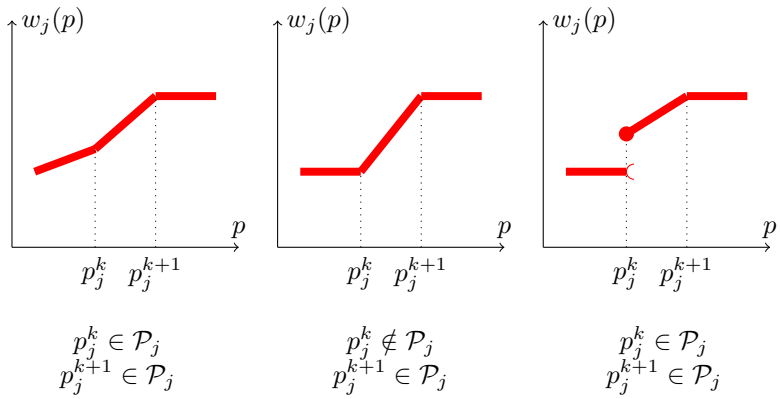


Figure 6: Examples of "interesting" and not "interesting" processing times.

is dominant.

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

Proof (1) have already been proven by Lemma 2.1.

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 2.4 is that when all tasks have the same deadline, there will be at most one singular on each machine:

Corollary 2.5. *For $P \mid PLP, d_j{=}d \mid -\sum w_j(p_j)$, the set of solutions such that:*

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

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

is dominant.

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

3. 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.

For one machine problems, we have:

Theorem 3.1. *$1 \mid d_j{=}d \mid -\sum w_j(p_j)$ is weakly NP-complete with the following profit functions:*

- *LPSTIP,* b_j=b
- *LBPST,* b_j=b
- *LBPSTIP,* b_j=b, $w_j^{\max}$=$w^{\max}$
- *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 2.3). We prove their NP-completeness using reductions from KNAPSACK, known to be NP-complete [8] 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 $\sum_{j \in S} v_j \geq V$?

For LPSTIP, $b_j = b$, 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.

For LBPST, b_j=b, 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 $1 \mid$ 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 2.5 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.

For LBPSTIP, b_j=b, $w_j^{\max}$=$w^{\max}$, 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 $1 \mid$ 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 2.5 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 $w(S') - w(S) \leq b_j d < B+1$. Therefore, $w(S') \geq (B+1)V$, and it is possible to build the corresponding solution for KNAPSACK. Therefore, I has a "yes" answer.

For LBPSTIP, $p_j^{\max}$=$p^{\max}$, $w_j^{\max}$=$w^{\max}$, 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 $1 \mid$ 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 2.5 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 3.2. *$P \mid d_j{=}d \mid -\sum w_j(p_j)$ 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:*

- *LPSTIP, $w_j^{\min}{=}w^{\min}$, $b_j{=}b$*
- *LBP, $w_j^{\max}{=}w^{\max}$*
- *LBP, $b_j{=}b$*
- *LBPST, $b_j{=}b$, $w_j^{\max}{=}w^{\max}$*

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 2.3 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 [8] and strongly NP-complete [9].

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?

For the first item, $Pm \mid\mid C_{\max}$ 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 as there are items, and for task T_j we set $w^{\min} = 1$, $p_j^{\min} = x_j$ and $d = \delta$.

For the second item, 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 $w(S) \geq n$? 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.

For the third item, we choose $b = 1$ and $w_j^{\max} = x_j$ and the question is: is there a schedule S such that $w(S) \geq \sum_{j=1}^{n} x_j$? The reasonning is the same.

For the last item, let I' be an instance of $1 \mid$ 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.

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

Corollary 3.3. *$P \mid PLP \mid -\sum w_j(p_j)$ is strongly NP-complete. $Pm \mid PLP \mid -\sum w_j(p_j)$ is weakly NP-complete.*

4. Polynomial algorithms for some piecewise-linear profit functions

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.

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 2.4, we can deduce the dominant set of possible start dates and end dates:

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

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

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

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 $|\{p_j^k\}| \leq \kappa$ to specify problems for which the number of different values for p_j^k is bounded by κ.

Then we can adapt Algorithm 1 to run on instants of Θ instead of $\{0, \ldots, d\}$:

Theorem 4.1. *Algorithm 2 returns an optimal solution to $Pm \mid PLP, |\{p_j^k\}| \leq \kappa \mid -\sum w_j(p_j)$ in polynomial time.*

Proof The proof is similar to the one of Theorem 2.3 in combination with Lemma 2.4 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. □

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 $k \neq i$.

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

Corollary 4.2. *$Pm \mid \mid -\sum w_j(p_j)$ 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:

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

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 $O(mn^{2m+3}K^{m+1} + n \log n)$.

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 4.3. *For $1 \mid PLP, b_j^k \in \{0, b\} \mid -\sum w_j(p_j)$, 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 2.1.

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 $T_1, \ldots, T_j$ (resp. $T_j, \ldots, T_n$) 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 4.4. *For $1 \mid PLP, b_j^k \in \{0,b\} \mid -\sum w_j(p_j)$, 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 4.3.

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

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

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

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

Let

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

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 4.5. *Algorithm 3 returns an optimal solution to* $1 \mid PLP, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0,b\} \mid -\sum w_j(p_j)$ *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 $(j,w) \in (\{0,\dots,n\} \times \Omega)$
3: **return**

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

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 4.6. $1 \mid \mid -\sum w_j(p_j)$ *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 4.4 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 3.2.

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 2.5 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 4.7. *For* $1 \mid PLP$, $b_j^k \in \{0, b_j\}$, 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 2.5.

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

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

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 4.8. *Algorithm 4 returns the optimal value of* $1 \mid PLP, |\{w_j^k\}| \leq \kappa, b_j^k \in \{0, b_j\}, d_j{=}d \mid -\sum w_j(p_j)$ *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 4.9. $1 \mid d_j{=}d \mid -\sum w_j(p_j)$ *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 4.7 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 3.2.

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
```

5. Conclusions

We proposed and defined a new category of scheduling problems which are of practical interest: processing time dependent profit 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, and we proved several cases to be polynomial and some others to be NP-complete, 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

[1] A.-M. Lagrange, P. Rubini, N. Brauner, H. Cambazard, N. Catusse, P. Lemaire, L. Baude, 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, 2016.

[2] N. Catusse, H. Cambazard, N. Brauner, P. Lemaire, B. Penz, A.-M. Lagrange, P. Rubini, A Branch-And-Price Algorithm for Scheduling Observations on a Telescope, in: Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), AAAI Press, 2016, pp. 3060–3066.

[3] J. K. Dey, J. Kurose, D. Towsley, On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks, IEEE Transactions on Computers 45 (1996) 802–813.

[4] D. Shabtay, G. Steiner, A survey of scheduling with controllable processing times, Discrete Applied Mathematics 155 (2007) 1643–1666.

[5] A. Shioura, N. V. Shakhlevich, V. A. Strusevich, Preemptive models of scheduling with controllable processing times and of scheduling with imprecise computation: A review of solution approaches, European Journal of Operational Research 266 (2018) 795–818.

[6] M. Sterna, A survey of scheduling problems with late work criteria, Omega 39 (2011) 120–129.

[7] M. L. Pinedo, Scheduling: theory, algorithms, and systems, Springer, 2016.

[8] M. R. Garey, D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, 1979.

[9] M. R. Garey, D. S. Johnson, Strong NP-Completeness Results: Motivation, Examples, and Implications, J. ACM 25 (1978) 499–508.