# Parallel machine scheduling with minimum number of tardy jobs: approximation and exponential algorithms

Federico Della Croce, Vincent t'Kindt, Olivier Ploton

# Parallel machine scheduling with minimum number of tardy jobs: approximation and exponential algorithms

Federico Della Croce[a], Vincent T'kindt[b,*], Olivier Ploton[c]

[a] *DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, CNR, IEIIT, Torino, Italy*
[b] *Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée (EA 6300), ERL CNRS 7002 ROOT, Tours, France, and DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy*
[c] *Université de Tours, Tours, France*

**Abstract**

We consider the solution of some $\mathcal{NP}$-hard parallel machine scheduling problems involving the minimization of the weighted or unweighted number of tardy jobs. We first show that these problems cannot be approximated in polynomial time. Then we propose exponential-time approximation algorithms and fixed parameter tractable exact algorithms to solve them.

*Keywords:* Number of tardy jobs, Identical Parallel Machines Scheduling, Exponential time approximation

## 1. Introduction

Scheduling theory deals with the allocation of a given set of jobs to resources over time while minimizing some criteria. Often, scheduling problems turn out to be $\mathcal{NP}$-hard and exact as well as heuristic algorithms have been proposed in the literature to solve them (Pinedo (2016)). Such algorithms are classically designed with the purpose of either being effective in practice or providing theoretical worst-case guarantees.

We mention the existence in the literature of *exact exponential time algorithms* (EETA) that are exact algorithms whose worst-case time complexity can be expressed as $\mathcal{O}^*(c^n)$ with $c$ a constant and $n$ is a measure of the input size (typically the number of jobs in the instance). The $\mathcal{O}^*$ notation suppresses polynomial factors in the complexity, *i.e.* $\mathcal{O}^*(c^n) = \mathcal{O}(p(n)c^n)$ where $p(n)$ is a polynomial on $n$. The proposal of such algorithms for $\mathcal{NP}$-hard scheduling problems started recently to be intensively considered in the literature. The survey by Woeginger (2003) revealed the existence of EETA for some single

*Corresponding author
*Email addresses:* federico.dellacroce@polito.it (Federico Della Croce), tkindt@univ-tours.fr (Vincent T'kindt), olivier.ploton@univ-tours.fr (Olivier Ploton)

machine scheduling problems. Later on, such algorithms have been proposed for a set of classic scheduling problems (Cygan et al., 2011; Garraffa et al., 2018; Lente et al., 2013; Lenté et al., 2014; Shang et al., 2018). Noteworthy, Lente et al. (2013) describe a generic *Sort & Search* approach enabling to derive EETA for scheduling problems. Under the *Exponential Time Hypothesis*, Jansen et al. (2013) present some results on the existence of lower bounds on the worst-case time complexities of some scheduling problems.
For some problems, it is possible to identify some parameter $\ell$ such that there exists an EETA to solve it in time $f(\ell)n^{O(1)}$, with $f$ being a function depending only on $\ell$: we say that the problem is fixed-parameter tractable (FPT, Downey and Fellows (1999)). This parameter can be problem dependent or related to the objective function value. Fixed parameter tractability provides highlights on the hardness of problems by exhibiting to which parameters it can be related. Recent FPT algorithms in scheduling theory were proposed by Mnich and Wiese (2015); Mnich and van Bevern (2018); Hermelin et al. (2018); Bessy and Giroudeau (2020). Most of the parameters used in these algorithms are related to the objective function value.

Approximation algorithms are particular heuristic algorithms that offer theoretical guarantees: they are proved to compute a solution whose distance to the optimal one is bounded. Let $Alg$ be the value of the solution computed by an approximation algorithm and $Opt$ be the optimal solution value. An approximation algorithm admits an approximation ratio $\rho$ if for any instance of the minimization problem $Alg \leq \rho Opt$. Additionally, if its running time is polynomial in the input size $n$, it is said to be a polynomial-time approximation algorithm. A *polynomial-time approximation scheme* (PTAS) is an approximation algorithm having $\rho = (1 + \epsilon)$, with $\epsilon > 0$, and that runs in polynomial time of $n$ when $\epsilon$ is fixed. A *fully polynomial-time approximation scheme* (FPTAS) is a particular PTAS whose running time is bounded by a polynomial of $n$ and $1/\epsilon$. Both PTAS and FPTAS have the advantage to be able to arbitrarily approach optimal solutions but at the price of a polynomial time complexity that increases for small values of $\epsilon$. Notice that some $\mathcal{NP}$-hard optimization problems do not admit polynomial-time approximation algorithms (unless $\mathcal{P}$=$\mathcal{NP}$). Approximation algorithms with fixed ratio $\rho$, PTAS or FPTAS have been largely considered in the scheduling literature, so that reporting them is out-of-the-scope of this paper.

Our work is at the intersection of EETA and approximation algorithms. How can we approach optimal solutions for $\mathcal{NP}$-hard problems that do not admit a polynomial-time approximation algorithm? Recent works on optimization problems have shown the opportunity of providing, in such a case, approximation algorithms with a moderately exponential worst-case time complexity (see for instance Paschos (2015)). Marx (2008) introduces the notion of fpt-approximation scheme (FPT-AS) that is an approximation algorithm computing, for a given $\epsilon > 0$, a solution with ratio $\rho = (1 + \epsilon)$ in time $f(\epsilon, \kappa) \cdot n^{O(1)}$, with some computable function $f$ and $\kappa$ a parameter of the instance. The latter is usually either a value associated with the instance, or a measure related to the objective function value. In this paper we focus on some parallel machines scheduling problems which cannot be approximated in polynomial time, thus leading to consider approximation in "exponential" time. Few works in the scheduling literature can be found on this topic, and they are mainly devoted to shop scheduling problems involving a set of $m$ machines and $n$ jobs to be scheduled. For an openshop problem,

Sevastianov and Woeginger (1998) propose a FPT-AS running in $f(\epsilon, m) + O(n \log(n))$ time, while for a flowshop problem Hall (1998) proposes one running in $f(\epsilon, m) + O(n^{3.5})$ time. Finally, Jansen et al. (2003) propose for a jobshop problem a FPT-AS running in $f(\epsilon, m, n_{max}) + O(n)$ time, with $n_{max}$ the maximum number of operations for a job.

We consider the problem of scheduling $n$ jobs on $m$ identical parallel machines. Each job $j$ is defined by a processing time $p_j$, a due date $d_j$ and it has to be processed non-preemptively by one of the machines. Each machine can process one job at a time. The aim is to compute a schedule so that the number of tardy jobs, denoted by $\sum_j U_j$, is minimized. For any given schedule $s$, let $C_j(s)$ be the completion time of job $j$: whenever $C_j(s) > d_j$ we set $U_j = 1$ (job is tardy), and $U_j = 0$ otherwise (job is early). Without loss of generality, we assume that $d_1 \leq d_2 \leq ... \leq d_n$. Using the standard three–field notation due to Graham et al. (1979) this problem is denoted by $P||\sum_j U_j$. If a single machine is available, *i.e.* $m = 1$, the problem can be solved in polynomial time Lawler and Moore (1969); Lin and Wang (2007). However, it turns out to be $\mathcal{NP}$-hard even in the case of two machines (Garey and Johnson, 1979). Leung and Yu (1994) proposed for a specific performance ratio and when $m = 2$, a polynomial-time approximation algorithm. With respect to the exact solution of the $m$ machine problem, Lente et al. (2013) proposed an EETA based on the *Sort & Search* technique. This algorithm requires $\mathcal{O}^*((m+1)^{\frac{n}{2}})$ time and space in the worst case. For instance, when $m = 2$, this complexity becomes $\mathcal{O}^*(3^{\frac{n}{2}}) = O(1.7321^n)$. Notice that this algorithm was originally designed for the weighted number of tardy jobs problems, denoted by $P||\sum_j w_j U_j$.

The remainder is organized as follows. Section 2 introduces preliminary results. Section 3 introduces a first exponential-time approximation algorithm based on a branching strategy. Section 4 shows how to improve the previous algorithm by adding a preprocessing step. As a side result, we show in section 5 that minimizing the number of tardy jobs is fixed-parameter tractable. Some extensions to related problems are discussed in section 6. Conclusions and future research directions are given in section 7.

## 2. Preliminary results

We first provide a result that rules out the existence of polynomial-time approximation schemes for the $P||\sum_j U_j$ problem.

**Theorem 1.** *The $P2||\sum_j U_j$ problem does not admit a polynomial time approximation algorithm with a bounded approximation ratio unless $\mathcal{P} = \mathcal{NP}$.*

Proof. The result is proved by reduction from the $P2|C_{max} \leq d|-$ problem, a well-known $\mathcal{NP}$-complete decision problem where $n$ jobs $j$ with processing times $p_j$ ($j = 1, \ldots, n$), two identical parallel machines and a value $d$ are given. This problem asks whether there exists a schedule of the jobs such that the makespan $C_{max}$ is at most $d$. We build an instance of $P2||\sum_j U_j$ by considering $n$ jobs with the processing times $p_j$ and due dates $d_j = d$ ($j = 1, \ldots, n$). This implies that an optimal solution of the $P2|d_j = d|\sum_j U_j$ instance attains value 0 only if $P2|C_{max} \leq d|-$ is feasible. Thus, if there was a polynomial-time approximation algorithm with a bounded approximation ratio $\rho > 1$ (even possibly as a function $\rho(n)$), we could decide $P2|C_{max} \leq d|-$ problem

just by checking if the approximate solution of $P2|d_j = d|\sum_j U_j$ is strictly positive. Clearly this is not possible under the assumption that $\mathcal{P} \neq \mathcal{NP}$.$\square$

Leung and Yu (1994) proposed an approximation result for the $P2||\sum_j U_j$ problem. As Theorem 1 holds, they defined the performance ratio as:

$$\rho_E = \frac{n - \sum_j U_j^*}{n - \sum_j U_j^H},$$

with $\sum_j U_j^*$ (resp. $\sum_j U_j^H$) the number of tardy jobs in the optimal solution (resp. the solution computed by their heuristic). They showed that the ratio $\rho_E = \frac{4}{3}$ is tight. In this paper, we consider the problem of approximating in exponential time the $P||\sum_j U_j$ problem by considering the classical approximation ratio:

$$\rho = \frac{\sum_j U_j^H}{\sum_j U_j^*}.$$

The existence of the EETA of Lente et al. (2013) creates the challenge of finding an exponential time approximation algorithm whose worst-case time complexity is lower than that of the EETA.

We now present a side result which will be used later on in our approximation algorithms. This result states the complexity of solving the decision variant of the identical parallel machine scheduling problem. This problem is denoted by $P|\tilde{d}_j|-$, with $\tilde{d}_j$ the deadline of job $j$. As it will appear in the remainder, improving the time complexity for solving this problem immediatly leads to improving the worst-case time complexity of the proposed exponential-time approximation algorithms.

**Theorem 2.** *Let $\tilde{d}_j$ be a deadline associated with job $j$, so that in a feasible schedule job $j$ must complete before $\tilde{d}_j$. The existence of a feasible schedule for the $P|\tilde{d}_j|-$ problem can be decided in $\mathcal{O}^*(m^{\frac{n}{2}})$ time and space.*

Proof. The $P|\tilde{d}_j|-$ problem is $\mathcal{NP}$-complete and we prove the result by showing the existence of an EETA based on the *Sort & Search* technique. Starting from the work of Lente et al. (2013) we need to reformulate the problem as a *multiple constraint problem* (MCP): this will imply the existence of a *Sort & Search* algorithm to solve the decision problem. We kindly refer the reader to Lente et al. (2013) for a detailed presentation of a MCP. Let us define $I_1 = \{1, ..., \lfloor \frac{n}{2} \rfloor\}$ and $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, ..., n\}$. With each partition $\mathcal{E}_1^k = (E_{1,1}^k, ..., E_{1,m}^k)$ of $I_1$ ($1 \leq k \leq m^{|I_1|}$), we associate a schedule $s_1^k$ in which jobs in $E_{1,\ell}^k$ are scheduled by increasing order of their index on machine $\ell$. We do the same to create the $m^{\frac{n}{2}}$ partitions $\mathcal{E}_2^i$ of $I_2$ and, correspondingly, to each partition $\mathcal{E}_2^i$ a schedule $s_2^i$ can be derived. A complete schedule for the original problem is then defined by the concatenation of a schedule $s_1^k$ and a schedule $s_2^i$, and is denoted by $s_1^k // s_2^i$.
Let us denote by $L_{max}^\ell(s|t)$ the value of the maximum lateness of sequence $s$ on machine $\ell$ when it starts at time $t$, *i.e.* $L_{max}^\ell(s|t) = L_{max}^\ell(s|0) + t$. We also have $L_{max}^\ell(s|t) = \max_{j \in s}(C_j^\ell(s|t) - \tilde{d}_j)$, with $C_j^\ell(s|t)$ the completion time of job $j$ in a schedule $s$ starting at time $t$ on machine $\ell$. Then, for any schedule $s_1^k // s_2^i$ we have:

$$L_{max}(s_1^k // s_2^i) = \max_{1 \leq \ell \leq m}(L_{max}^\ell(s_1^{k,\ell}|0); L_{max}^\ell(s_2^{i,\ell}|C_{max}^\ell(s_1^{k,\ell}))),$$

with $C^{\ell}_{max}(s)$ the makespan of schedule $s$ on machine $\ell$, and $s^{a,\ell}_b$ the sequence of jobs on machine $\ell$ in schedule $s^a_b$. Schedule $s^k_1//s^i_2$ is feasible with respect to the deadlines if and only if $L_{max}(s^k_1//s^i_2) \leq 0$.

With each of the $m^{\frac{n}{2}}$ schedules $s^k_1$ (resp. $s^i_2$) generated from $I_1$ (resp. $I_2$) we associate a vector $\vec{a}_k$ (resp. $\vec{b}_i$). The $P|\tilde{d_j}|-$ problem can be reformulated as the following MCP:

$$\begin{array}{l} \text{Minimize } f(\vec{a_k}, b^0_i) \\ \text{s.t.} \\ \quad g_\ell(\vec{a_k}, b^\ell_i) \geq 0, \quad (1 \leq \ell \leq m) \\ \quad \vec{a}_k \in A,\, \vec{b}_i \in B. \end{array}$$

with
$$\begin{cases} A & \text{the set of all vectors } \vec{a}_k \text{ associated with the } \mathcal{E}^k_1\text{'s} \\ B & \text{the set of all vectors } \vec{b}_i \text{ associated with the } \mathcal{E}^i_2\text{'s} \\ \vec{a}_k = & (L_{max}(s^k_1),\ C_{max}(s^{k,1}_1),\ ...,\ C_{max}(s^{k,m}_1)) \\ \vec{b}_i = & (0,\ L_{max}(s^{i,1}_2|0),\ ...,\ L_{max}(s^{i,m}_2|0)) \\ f(\vec{a}_k, b^0_i) = & L_{max}(s^k_1) \\ g_\ell(\vec{a}_k, b^\ell_i) = & -C_{max}(s^{k,\ell}_1) - L_{max}(s^{i,\ell}_2|0) \end{cases}$$

Following Lente et al. (2013), there exists an EETA algorithm requiring $\mathcal{O}^*(m^{\frac{n}{2}})$ time and space to solve this problem. The algorithm returns a solution whose objective function value is not positive if and only if there exists a feasible schedule with respect to the deadlines.□

## 3. A branching based approximation algorithm

Branching algorithms are known to be usable for deriving approximation algorithms with a moderate worst-case time complexity (*e.g.* Escoffier et al. (2016)). In this section, we show that, by decomposing the problem solution and introducing a parametrized branching scheme, we can derive an approximation algorithm, referred to as `Bapprox`, for the $P||\sum_j U_j$ problem.

Let $k > 0$ be an integer parameter and jobs be grouped into $\lceil \frac{n}{k} \rceil$ batches. Each batch $B_\ell$ contains jobs $\{(\ell-1)k+1, ..., \ell k\}$, $1 \leq \ell \leq \lfloor \frac{n}{k} \rfloor$, and there exists a last batch $B_{\lceil \frac{n}{k} \rceil}$ containing the last $(n - k\lfloor \frac{n}{k} \rfloor)$ jobs if $\frac{n}{k}$ is not integral. Algorithm `Bapprox` builds a binary search tree by branching at each level $k$ on batch $B_k$ and scheduling all its jobs either early or tardy (Figure 1).

Then, for each leaf node we have a set of tardy jobs and a set of early jobs. The existence of a feasible schedule for the latter is tested in $\mathcal{O}^*(m^{\frac{n}{2}})$ time by solving the corresponding $P|\tilde{d_j} = d_j|-$ problem. If such a schedule exists then all the tardy jobs are scheduled on any machine after the early jobs. Algorithm `Bapprox` is given in Figure 2. We now state its worst-case ratio and complexities in Propositions 1 and 2.

**Proposition 1.** *Algorithm `Bapprox` admits a worst-case ratio $\rho \leq k$.*

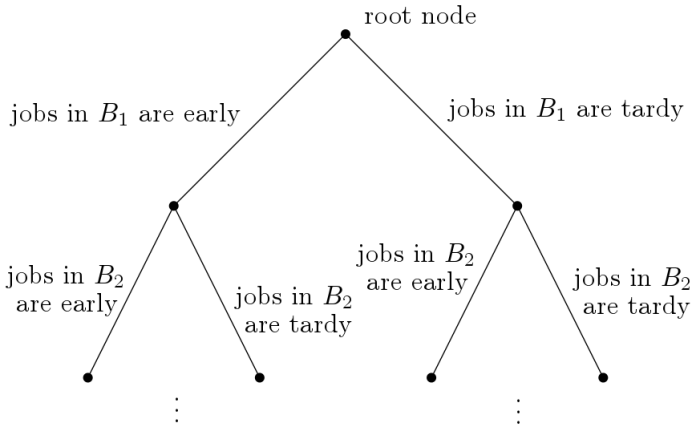


Figure 1: Illustration of the search tree

Algorithm `Bapprox`

**Input:** An instance $I$ with $n$ jobs and $m$ machines, a parameter $k$.
**Output:** A schedule $s^B$ with objective function value $\sum_j U_j^B$.
Solve the $P|\tilde{d}_j = d_j|-$ problem: if there exists a feasible schedule $s$, **return** $s$ and $\sum_j U_j(s) = 0$.
$s^B = \emptyset$, $\sum_j U_j^B = +\infty$.
$B_\ell = \{(\ell-1)k+1, ..., \ell k\}$, $\forall \ell = 1...\lfloor \frac{n}{k} \rfloor$.
**if** ($\frac{n}{k}$ is not integral) **then**
    $B_{\lceil \frac{n}{k} \rceil} = \{k\lfloor \frac{n}{k} \rfloor, ..., n\}$.
**end if**
$\mathcal{LN} = \{(E_u; T_u) : E_u, T_u \subseteq \{B_1, ..., B_{\lceil \frac{n}{k} \rceil}\}, E_u \cup T_u = I, E_u \cap T_u = \emptyset\}$
**for** $((E;T) \in \mathcal{LN})$ **do**
    Solve the $P|\tilde{d}_j = d_j, j \in E|-$ problem: Let $s_E$ be a feasible schedule if it exists.
    Let $s_T$ be a schedule in which jobs in $T$ are in any order on the first machine.
    $s = s_E // s_T$.
    **if** ($s_E$ is feasible and $|T| < \sum_j U_j^B$) **then**
        $s^B = s$, $\sum_j U_j^B = |T|$.
    **end if**
**end for**
**return** $s^B$ and $\sum_j U_j^B$.

Figure 2: A branching based approximation algorithm for the $P||\sum_j U_j$ problem

PROOF. First notice that if there exists a feasible schedule with no tardy jobs, then this one is found at step 2 and `Bapprox` is optimal.
So, let us assume that in the optimal schedule there is at least one tardy job. In the schedule $s^B$, returned by `Bapprox`, assume that there are $\alpha$ batches $B_j^B$ which are scheduled tardy. Let $E^B$ be the set of early jobs in $s^B$. We claim that, $\forall j = 1..\alpha$, there is no optimal solution in which all jobs in $E^B \cup B_j^B$ can be scheduled early together. First observe that, if there exists an optimal solution in which all jobs $E^B$ are early, then scheduling early all jobs of a batch $B_j^B$ implies a contradiction with the fact that $s^B$ is the best solution found by algorithm `Bapprox`. This implies that, at least one job from $B_j^B$ is tardy in an optimal solution in which all jobs in $E^B$ are early. So in the other case, at least one job $u \in E^B$ must be scheduled tardy to obtain an optimal solution. Besides, scheduling $u$ early in `Bapprox` disables some batch $B_j^B$ from being scheduled early. Notice that, due to the branching scheme, we cannot have one such early job $u$ preventing two batches $B_j^B$ and $B_{j'}^B$ from being early: otherwise the branch in which these two batches are scheduled early and the batch containing $u$ is scheduled tardy would be feasible and better than $s^B$.
Let $\ell_j$ be the number of jobs that disable jobs in batch $B_j^B$ from being early: to obtain an optimal solution, all these $\ell_j$'s jobs should be scheduled tardy and all other tardy jobs in $s^B$ should be scheduled early. Following the previous analysis, we can derive that $1 \leq \ell_j \leq k$. Then,

$$\rho = \frac{\alpha k}{\sum_j \ell_j},$$

with $\sum_j \ell_j \geq \alpha$. The ratio is maximum whenever $\sum_j \ell_j = \alpha$, leading to $\rho \leq k$.□

**Remark 1.** *The ratio $k$ stated in Proposition 1 is tight. Consider a 2 machines and 4 jobs example with: $p_1 = 5, p_2 = 5, p_3 = 4, p_4 = 6$ and $d_1 = 7, d_2 = 8, d_3 = 9, d_4 = 10$ and $k = 2$. An optimal solution has 1 job tardy, e.g. jobs 1 and 3 on machine 1 and jobs 2 and 4 on machine 2. Algorithm `Bapprox` generates a solution $s^B$ with 1 batch of early jobs (e.g. $\{1; 2\}$) and 1 batch of tardy jobs (e.g. $\{3; 4\}$). So, the number of tardy jobs in $s^B$ is equal to $k = 2$.*

**Proposition 2.** *Algorithm `Bapprox` requires $\mathcal{O}^*((1 + m^{\frac{k}{2}})^{\frac{n}{k}})$ time and $\mathcal{O}^*(m^{\frac{n}{2}})$ space.*

PROOF. We first state the worst-case time complexity. Step 2 requires $\mathcal{O}^*(m^{\frac{n}{2}})$ time. Steps 3-7 are processed in polynomial time of $n$.
At step 8, leaf nodes contained in set $\mathcal{LN}$ correspond to all possible selections of $\ell$ early batches over $\lceil \frac{n}{k} \rceil$, and so $|\mathcal{LN}| = \sum_{\ell=0}^{\lceil \frac{n}{k} \rceil} \binom{\lceil \frac{n}{k} \rceil}{\ell}$. For each $(E; T) \in \mathcal{LN}$, step 10 requires $\mathcal{O}^*(m^{\frac{|E|}{2}})$ time, with $|E| = k\ell$ and $\ell$ the number of early batches in $E$. It follows that steps 8-16 have a worst-case running time in:

$$\mathcal{O}^*(\sum_{\ell=0}^{\lceil \frac{n}{k} \rceil} \binom{\lceil \frac{n}{k} \rceil}{\ell} (m^{\frac{k}{2}})^{\ell}) = \mathcal{O}^*((1 + m^{\frac{k}{2}})^{\frac{n}{k}}),$$

by making use of the Newton's binomial formula.
Concerning the worst-case space requirement of algorithm `Bapprox`, step 2 requires $\mathcal{O}^*(m^{\frac{n}{2}})$ space. By exploring the search tree following a depth first search strategy, algorithm `Bapprox` can be implemented in such a way that each time a leaf node is found the corresponding $P|\tilde{d}_j = d_j, j \in E|-$ problem is directly solved. This implies that steps 8-16 can be implemented in $\mathcal{O}^*(m^{\frac{n}{2}})$ space. □

To illustrate the above propositions, we provide in Table 1 values of ratios and complexities on the two-machine problem. Noteworthy, by comparison to the EETA running in $O(1.7320^n)$ time, algorithm `Bapprox` is relevant for $k \geq 3$.

| $k$ | $\rho$ | time |
|---|---|---|
| 1 | 1 | $O(2.4142^n)$ |
| 2 | 2 | $O(1.7320^n)$ |
| 3 | 3 | $O(1.5643^n)$ |
| 4 | 4 | $O(1.4953^n)$ |
| 5 | 5 | $O(1.4610^n)$ |
| | | ... |
| 10 | 10 | $O(1.4186^n)$ |

Table 1: Some ratios and complexities for the algorithm `Bapprox` on the $P2||\sum_j U_j$ problem

## 4. Branching with preprocessing

In this section we consider a second exponential-time approximation algorithm, referred to as `PBapprox`, based on algorithm `Bapprox`. The idea is to include a preprocessing step before running `Bapprox` in order to decrease the approximation ratio. This is in the spirit of the memoization techniques applied in graph optimization problems (see, e.g. Robson (1986); Fomin et al. (2009)). Let us introduce a parameter $c$ which is the fraction of the jobset to be considered by the preprocessing. This one generates all possible subsets of at most $\frac{n}{c}$ tardy jobs and then, for each, solves the $P|\tilde{d}_j = d_j|-$ problem on the remaining jobs. If at least one of these subsets leads to a feasible schedule, then the optimal solution of the $P||\sum_j U_j$ problem is found. Otherwise, algorithm `Bapprox` is used. Algorithm `PBapprox` is given in Figure 3. We state its worst-case ratio and complexities in Propositions 3 and 4.

**Proposition 3.** *Algorithm `PBapprox` admits a worst-case ratio* $\rho \leq \frac{k^2+k(c-1)+1}{k+c}$.

Proof. If `PBapprox` stops in steps 2-13 then it has found the optimal solution. So, the approximation ratio is obtained when running algorithm `Bapprox`.
Let $s^{PB} = s^B // s_T$ be the solution returned by algorithm `PBapprox`. Following the analysis done in proof of Proposition 1, we assume that in $s^B$, $\alpha \leq \lceil \frac{n(c-1)}{ck} \rceil$ batches have been scheduled tardy. Assume that in the optimal solution of the problem restricted to the jobs in $R = \{i/i \in s^B\}$, we have $\mu$ tardy jobs. Then, in the worst case it may happen that $\alpha = \mu$. So, we derive:

$$\rho \leq \frac{k \min(\mu; \lceil \frac{n(c-1)}{ck} \rceil) + \frac{n}{c}}{\mu + \frac{n}{c}}.$$

Two cases can occur. In case 1, $\mu > \lceil \frac{n(c-1)}{ck} \rceil$ and then:

$$\rho \leq \frac{k(n(c-1)+ck)+n}{n(c-1)+ck+n}.$$

Algorithm `PBapprox`

**Input:** An instance $I$ with $n$ jobs and $m$ machines, parameters $k$ and $c$.
**Output:** A schedule $s^B$ with objective function value $\sum_j U_j^B$.
Solve the $P|\tilde{d}_j = d_j|-$ problem: if there exists a feasible schedule $s$, **return** $s$ and $\sum_j U_j(s) = 0$.
$s^{PB} = \emptyset$, $\sum_j U_j^{PB} = +\infty$.
// Preprocessing step
**for** ($i = 1$ to $\lfloor \frac{n}{c} \rfloor$) **do**
  $\mathcal{S}_i = \{$subsets of $i$ tardy jobs$\}$.
  **for** ($T \in \mathcal{S}_i$) **do**
    Solve the $P|\tilde{d}_j = d_j, j \in I \backslash T|-$ problem: Let $s_E$ be a feasible schedule if it exists.
    **if** ($s_E$ is feasible) **then**
      Let $s_T$ be a schedule in which jobs in $T$ are in any order on the first machine.
      **return** $s^{PB} = s_E // s_T$ and $\sum_j U_j^{PB} = i$.
    **end if**
  **end for**
**end for**
// No solution has been found: using algorithm `Bapprox`
**for** ($T \in \mathcal{S}_{\lfloor \frac{n}{c} \rfloor}$) **do**
  $(s^B, \sum_j U_j^B)$ =`BApprox`($I \backslash T$, $n - \lfloor \frac{n}{c} \rfloor$,$m$,$k$).
  Let $s_T$ be a schedule in which jobs in $T$ are in any order on the first machine.
  **if** $((\sum_j U_j^B + \lfloor \frac{n}{c} \rfloor) < \sum_j U_j^{PB})$ **then**
    $s^{PB} = s^B // s_T$, $\sum_j U_j^{PB} = \sum_j U_j^B + \lfloor \frac{n}{c} \rfloor$.
  **end if**
**end for**
**return** $s^B$ and $\sum_j U_j^B$.

Figure 3: A branching and preprocessing based approximation algorithm for the $P||\sum_j U_j$ problem

In order to impose the preprocesssing step to be executed we must have $n \geq c$. Besides, when $k$ and $c$ are fixed, the function:

$$g(n) = \frac{k(n(c-1)+ck)+n}{n(c-1)+ck+n},$$

is a decreasing function. Then, the ratio is maximum when $n = c$, leading to:

$$\rho \leq \frac{k^2+k(c-1)+1}{k+c} = \rho_1.$$

In case 2, $\mu \leq \lceil \frac{n(c-1)}{ck} \rceil$ and then we have:

$$\rho \leq \frac{ck\mu+n}{c\mu+n} = f(\mu).$$

As $f(\mu)$ is a non-decreasing function of $\mu$, it is maximum when $\mu = \frac{n(c-1)}{ck}$, leading to:

$$\rho \leq \frac{ck}{k+c-1} = \rho_2.$$

By analytically comparing $\rho_1$ and $\rho_2$ we can derive that $\rho_1 \geq \rho_2$ whenever $k \geq 1$ and $c \geq 1$. □

Before, establishing a bound on the worst-case time complexity, let us consider the partial sum of binomials $\sum_{i=0}^{\ell} \binom{n}{i} x^i y^{n-i}$, $x, y \in \mathcal{R}$. There is no closed formula to this partial sum even when $x = y = 1$ (Boardman (2004)). We provide the following result, whose proof is given in the appendix.

**Lemma 3.** *Let $c$ be an integer with $c \geq 2$. We have the following tight $\mathcal{O}^*$ bounds:*

$$\binom{n}{\lfloor \frac{n}{c} \rfloor} = \mathcal{O}^*\left(2^{H(\frac{1}{c})n}\right)$$

*and*

$$\sum_{i=0}^{\lfloor \frac{n}{c} \rfloor} \binom{n}{i} = \mathcal{O}^*\left(2^{H(\frac{1}{c})n}\right)$$

*with $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$, $0<x<1$, the binary entropy of $x$.*

To the best of our knowledge, the binary entropy in scheduling theory has been only used by Gawiejnowicz and Kurc (2020). We use it to establish an upper bound on the worst-case time and space complexities of `PBapprox`.

**Proposition 4.** *Algorithm `PBapprox` requires $\mathcal{O}^*\left(\left(2^{H(\frac{1}{c})}(1+m^{\frac{k}{2}})^{\frac{(c-1)}{ck}}\right)^n\right)$ time. The worst-case space requirement is in $\mathcal{O}^*\left(\left(m^{\frac{c-1}{2c}}\right)^n\right)$.*

Proof. We first prove the worst-case time complexity. Step 2 requires $\mathcal{O}^*(m^{\frac{n}{2}})$ time. The preprocessing phase (steps 4-13) involves the generation of subsets of size at most $\lfloor \frac{n}{c} \rfloor$ and the solution of feasibility problems. Then, this phase has the following worst-case running time:

$$\mathcal{O}^*(\textstyle\sum_{i=0}^{\lfloor \frac{n}{c} \rfloor} \binom{n}{i} m^{\frac{n-i}{2}}).$$

Following Lemma 3, we have the following upper bound:

$$\sum_{i=0}^{\lfloor \frac{n}{c} \rfloor} \binom{n}{i} m^{\frac{n-i}{2}} \leq \sum_{i=0}^{\lfloor \frac{n}{c} \rfloor} \binom{n}{i} \times m^{\frac{n(c-1)}{2c}} = \mathcal{O}^* \left( 2^{H(\frac{1}{c})n} m^{\frac{n(c-1)}{2c}} \right)$$

Therefore, the preprocessing phase has a worst-case time complexity in $\mathcal{O}^*(2^{H(\frac{1}{c})n} m^{\frac{n(c-1)}{2c}})$.
Now, let us consider the branching phase (steps 14-20). For each jobset of size $\lfloor \frac{n}{c} \rfloor$, algorithm `Bapprox` requires $\mathcal{O}^*((1+m^{\frac{k}{2}})^{\frac{n-\lfloor \frac{n}{c} \rfloor}{k}})$ time. Thus, the branching phase has a worst-case running time in:

$$\mathcal{O}^*(\binom{n}{\lfloor \frac{n}{c} \rfloor}(1+m^{\frac{k}{2}})^{\frac{n-\lfloor \frac{n}{c} \rfloor}{k}}).$$

Again, following Lemma 3, and knowing that $\forall x, x^{n+O(1)} = \mathcal{O}^*(x^n)$, we obtain:

$$\begin{aligned} \binom{n}{\lfloor \frac{n}{c} \rfloor}(1+m^{\frac{k}{2}})^{\frac{n-\lfloor \frac{n}{c} \rfloor}{k}} &= \mathcal{O}^* \left( 2^{H(\frac{1}{c})n} (1+m^{\frac{k}{2}})^{\frac{n-\lfloor \frac{n}{c} \rfloor}{k}} \right) \\ &= \mathcal{O}^* \left( 2^{H(\frac{1}{c})n} (1+m^{\frac{k}{2}})^{\frac{n-\frac{n}{c}}{k}} \right) \end{aligned}$$

Therefore, the worst-case time complexity is in

$$\mathcal{O}^* \left( \max \left( \left( 2^{H(\frac{1}{c})} (m^{\frac{1}{2}})^{\frac{(c-1)}{c}} \right)^n ; \left( 2^{H(\frac{1}{c})} ((1+m^{\frac{k}{2}})^{\frac{1}{k}})^{\frac{(c-1)}{c}} \right)^n \right) \right)$$

We have: $m^{\frac{k}{2}} < 1 + m^{\frac{k}{2}}$, so $m^{\frac{1}{2}} = (m^{\frac{k}{2}})^{\frac{1}{k}} < (1+m^{\frac{k}{2}})^{\frac{1}{k}}$, so the first term (preprocessing phase) is smaller and negligible, provided that $n$ is large enough.
Finally, the worst-case time complexity is in $\mathcal{O}^* \left( \left( 2^{H(\frac{1}{c})} ((1+m^{\frac{k}{2}})^{\frac{1}{k}})^{\frac{(c-1)}{c}} \right)^n \right)$

Following a similar reasoning than in the proof of Proposition 2, there exists an implementation of algorithm `PBapprox` in which both the preprocessing and the branching phases requires $\mathcal{O}^*(m^{\frac{n(c-1)}{2c}})$ space. □

To illustrate the above propositions, we provide in Table 2 some interesting values of ratios and complexities on the two-machine problem.

## 5. Fixed parameter tractability

We revisit the $Pm||\sum_j U_j$ problem, *i.e.* for a fixed number of machines $m$, under the fixed parameter tractability setting. We show that it is FPT if we take as a parameter the number of early jobs denoted by $|E|$. Notice that this parameter is related to the number of jobs in the instance and not to the values of jobs characteristics, which makes it an interesting parameter.

To this extent, consider the findings of Lin and Wang (2007) where it was shown that an alternative exact solution of problem $1|d_j|\sum_j U_j$ can be obtained by means of the algorithm `Moore-rev` given in Figure 4. Remember that it is assumed that jobs are

| $k$ | $c$ | $\rho$ | time |
|---|---|---|---|
| 1 | 100000 | 1 | $O(2.4145^n)$ |
| | 1000 | 1 | $O(2.4312^n)$ |
| | 100 | 1 | $O(2.5309^n)$ |
| 2 | 100000 | 1.99 | $O(1.7323^n)$ |
| | 1000 | 1.99 | $O(1.7448^n)$ |
| | 100 | 1.99 | $O(1.8218^n)$ |
| 3 | 1000 | 2.99 | $O(1.5761^n)$ |
| | 100 | 2.98 | $O(1.6471^n)$ |
| | 10 | 2.84 | $O(2.0706^n)$ |
| 4 | 1000 | 3.99 | $O(1.5066^n)$ |
| | 100 | 3.97 | $O(1.5751^n)$ |
| | 10 | 3.78 | $O(1.9882^n)$ |
| | | ... | |

Table 2: Some ratios and complexities for the algorithm `PBapprox` on the $P2||\sum_j U_j$ problem

indexed such that $d_1 \leq d_2 \leq ... \leq d_n$. The rationale of this algorithm is that, if the jobs are selected one at a time according to the non-decreasing order of their processing time, then, they are immediately determined to be early or tardy: if a job $j$ and the current set of early jobs can be all scheduled early (following the earliest due date rule), then job $j$ is early in an optimal solution. Otherwise, it is tardy.

The parameterized exact algorithm we propose, referred to as algorithm $\texttt{FPT}_E$, extends algorithm `Moore-rev` to the case of $m$ parallel machines. It explores a search tree by branching on the jobs in non-decreasing order of their processing time. For a given node of the search tree, the unscheduled job $j$ with the smallest processing time is tested on each machine $i$. If, as in algorithm `Moore-rev`, all the current early jobs $E_i$ on machine $i$ and job $j$ can be scheduled early, then a child node is created in which $E_i = E_i \cup \{j\}$. If on all machines job $j$ induces at least one tardy job, then it is scheduled tardy and the next unscheduled job is examined. Algorithm $\texttt{FPT}_E$ is described in Figure 5. Algorithm $\texttt{FPT}_E$ is optimal since, at each level of the search tree and for a node $(N, T, [E_u]_{u=1..m})$, the smallest unscheduled job $j$ is tested on all machines. If there is no machine on which it can be scheduled early, due to the result of Lin and Wang (2007), then it is necessarily tardy in a solution having jobs in the $E_u$'s early. Besides, if there is at least one machine $i$ such that $E_i \cup \{j\}$ can be scheduled early, then following Lin and Wang (2007) there is no gain in scheduling $j$ tardy on that machine (or on any machine).

**Proposition 5.** *Algorithm $\texttt{FPT}_E$ requires $\mathcal{O}^*\left(m^{|E|}\right)$ time and polynomial space, with $|E|$ the number of early jobs.*

Proof. At each feasible branch a job is scheduled early on a machine. In the worst-case, for each early job, at most $m$ branches are created. Besides, we know that if the optimal solution has $|E|$ early jobs, the search tree has at most $|E|$ levels. Correspondingly, the time complexity is $\mathcal{O}^*\left(m^{|E|}\right)$. For the space complexity, by visiting the search tree in

Algorithm `Moore-rev`

**Input:** An instance $I$ with $n$ jobs and 1 machine.
**Ouput:** The set of early jobs $E$ and the number of tardy jobs $\sum_j U_j$.
$N_{EDD} = \{1, 2, ..., n\}$ and $E = T = \emptyset$.
**while** $(N_{EDD} \neq \emptyset)$ **do**
  Choose a job $j \in N_{EDD}$ with the smallest processing time $p_j$.
  $N_{EDD} = N_{EDD} \setminus \{j\}$.
  **if** $(E \cup \{j\}$ is on-time) **then**
    $E = E \cup \{j\}$.
  **else**
    $T = T \cup \{j\}$.
  **end if**
**end while**
**return** $E$ and $\sum_j U_j = |T|$.

Figure 4: Moore's algorithm revised for the $1|d_j|\sum_j U_j$ problem

depth first, at most $(|E|+m)$ branches will be present at the same time inducing a spatial complexity polynomial in $n$. □

## 6. Extensions to related problems

In this section, we examine possible extensions of the previous algorithms to related problems. First, let us consider the problem where machines are unrelated, *i.e.* the processing time of job $j$ when processed on machine $i$ is $p_{j,i}$ $(j = 1, ..., n,\ i = 1, ..., m)$. Using the standard three-field notation, this problem is denoted by $R|d_j|\sum_j U_j$.
Algorithms `Bapprox` and `PBapprox` rely on the solution of decision problems. Then, to make them solving the $R|d_j|\sum_j U_j$ problem we need to consider the solution of the $R|\tilde{d}_j|-$ problem. The proof of Theorem 2 reveals that the proposed EETA also solves the $R|\tilde{d}_j|-$ problem in $\mathcal{O}^*(m^{\frac{n}{2}})$ time and space. Since, in the unrelated parallel machine problem, when the jobs are assigned to a machine the processing times are fixed, this algorithm directly applies. Consequently, Propositions 1, 2, 3 and 4 are valid for the $R|d_j|\sum_j U_j$ problem.
Let us now turn to the parameterized algorithm `FPT`. It is possible to slightly modify it in order to achieve the same results as those reported in Proposition 5. The corresponding algorithm $\texttt{FPT}^R$, proceeds as algorithm `FPT` but manipulates at each step operations $(j, i)$, $1 \leq j \leq n$, $1 \leq i \leq m$, instead of jobs $j$. Each time an operation $(j, i)$ is scheduled, all other operations of job $j$ are removed from the list of unscheduled operations. The optimal solution is given by the leaf node having the largest number of early jobs $|E|$, with $E = \cup_{i=1,...,m} E_i$.

Algorithm FPT

**Input:** An instance $I$ with $n$ jobs and $m$ machines.

**Ouput:** A schedule and its associated number of tardy jobs.

**Root node:**

$N = I$. // The set of unscheduled jobs

$T = \emptyset$. // The set of tardy jobs

$E_u = \emptyset, \forall u = 1, ..., m$. // The set of early jobs on each machine

**Branching rule:** for a given unexplored node $(N, T, [E_u]_{u=1..m})$,

$\mathcal{CN} = \emptyset$. // The list of children nodes

**while** ($\mathcal{CN} = \emptyset$ and $N \neq \emptyset$)

$j = argmin_{\ell \in N}(p_\ell)$.

**for** ($i = 1...m$) **do**

**if** ($E_i \cup \{j\}$ is on-time) **then**

**Create a child node:**

$N' = N \setminus \{j\}$, $T' = T$, and $E'_u = E_u, \forall u = 1, ..., m$.

$E'_i = E'_i \cup \{j\}$.

$\mathcal{CN} = \mathcal{CN} \cup \{(N', T', [E'_u]_{u=1..m})\}$

**end if**

**end for**

**if** ($\mathcal{CN} = \emptyset$) **then**

$N = N \setminus \{j\}$ and $T = T \cup \{j\}$.

**end if**

**end while**

Add $\mathcal{CN}$ to the list of unexplored nodes.

**Search strategy:** apply depth first strategy.

Figure 5: A FPT exact algorithm for problem $Pm|d_j|\sum_j U_j$

Notice that, to ease the presentations of `BApprox`, `PBApprox` and `FPT`, we have introduced them in the context of identical parallel machines to avoid heavier notations induced by manipulating operations instead of jobs.

Now, let us consider the weighted case. Each job $j$ has a weight $w_j$ and the objective turns to minimize the weighted number of jobs $\sum_j w_j U_j$. The corresponding problem is denoted by $P||\sum_j w_j U_j$.
Algorithm `Bapprox`, as described in Figure 2, does not admit a bounded ratio $\rho$. However, let `Bapprox`$^w$ be the variant of `Bapprox` in which the following modifications are done:

- Jobs are re-indexed such that $w_1 \geq w_2 \geq ... \geq w_n$.
- At any level of the level of the search three, if $j$ is the unscheduled job with the smallest index, then two branches are created: one in which job $j$ is scheduled early, and the other in which the jobs $\{j, ...j+k-1\}$ are scheduled tardy (Figure 6).

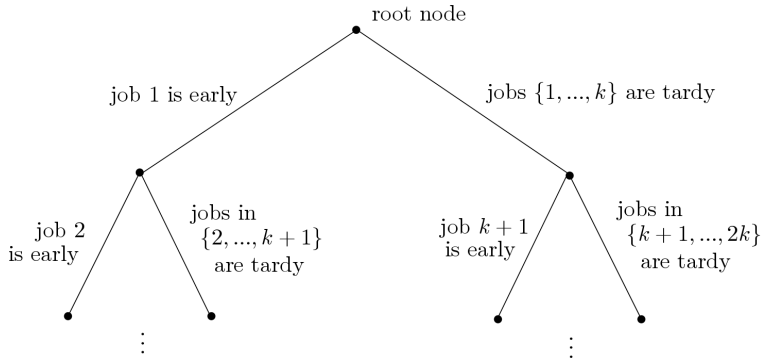


Figure 6: Illustration of the search tree (weighted case)

We can state the following propositions.

**Proposition 6.** *Algorithm* `Bapprox`$^w$ *admits a worst-case ratio* $\rho \leq k$.

Proof. As in the proof of Proposition 1, let us focus in the case where there is at least one tardy job.
In the schedule $s^B$ assume that there are $\alpha$ sets $\{\ell_u, ..., \ell_u + k\}$ which are scheduled tardy. In the worst-case scenario it can be assumed that, in the optimal solution, only one job for each of these sets is tardy. Due to the branching, these ones are necessarily jobs $\ell_1, \ell_2 ..., \ell_\alpha$. Then, as the jobs are indexed by non-increasing value of their weight, we can write:

$$\begin{aligned} \rho &\leq \frac{(w_{\ell_1}+...w_{\ell_1+k})+...+(w_{\ell_\alpha}+...w_{\ell_\alpha+k})}{w_{\ell_1}+...+w_{\ell_\alpha}} \\ &\leq \frac{k(w_{\ell_1}+w_{\ell_2}+...+w_{\ell_\alpha})}{w_{\ell_1}+...+w_{\ell_\alpha}} \\ &\leq k. \end{aligned}$$ □

**Proposition 7.** *Algorithm* `Bapprox`$^w$ *requires* $\mathcal{O}^*(\gamma^n)$ *time and* $\mathcal{O}^*(m^{\frac{n}{2}})$ *space, with* $\gamma = m^{\frac{1}{2\delta}}$ *and* $\gamma^{-k} + \gamma^{-1+\delta} = 1$.

PROOF. In the analysis of the running time of the algorithm above, we adopt a measure and conquer approach (see for instance Fomin et al. (2009)). More precisely, we do not count in the measure the jobs scheduled to be tardy (they receive weight 0), and we count with a weight $0 \leq \delta \leq 1$ the jobs scheduled to be early. Finally the unscheduled jobs receive weight 1. We so get recurrences on the time $T(p)$ required to solve instances of weight $p$, where the weight of an instance is the sum of the weights of its jobs. As in `Bapprox`, the search tree first decides whether jobs are early or tardy; then, once this decision has been taken on all jobs, solves the related $P|\tilde{d}_j = d_j, j \in E|-$ problem. This last step, as all the jobs have weight $\delta$ has a complexity $\gamma^n$ where

$$\gamma = m^{\frac{1}{2\delta}} \tag{1}$$

corresponding to a recursion of the type $T(p) \leq mT(p - 2\delta)$. Besides, at each node of the search tree, the branching rule creates two children nodes setting either one job early (whose weight decrease from 1 to $1 - \delta$) or $k$ jobs tardy. Thus, we have in this case $T(p) \leq T(p - (1 - \delta)) + T(p - k)$. This last recursion corresponds to the equality

$$\gamma^{-k} + \gamma^{-1+\delta} = 1. \tag{2}$$

Hence, for any given $m$, Algorithm `Bapprox`$^w$ runs with complexity $\mathcal{O}^*(\gamma^n)$ with $\gamma$ respecting both equations (1) and (2). For the space complexity, the same analysis applied in Proposition 1 to Algorithm `Bapprox` holds. Correspondingly, the space complexity is in $\mathcal{O}^*(m^{\frac{n}{2}})$.□

To illustrate the above propositions, we provide in Table 3 values of ratios, complexities and $\delta$ on the two-machine problem. Noteworthy, by comparison to the EETA running in $O(1.7320^n)$ time, this table shows that algorithm `Bapprox`$^w$ is relevant for $k \geq 4$.

| $k$ | $\rho$ | $\delta$ | time |
|---|---|---|---|
| 1 | 1 | 0.3923 | $O(2.4142^n)$ |
| 2 | 2 | 0.5263 | $O(1.9319^n)$ |
| 3 | 3 | 0.6236 | $O(1.7453^n)$ |
| 4 | 4 | 0.7002 | $O(1.6407^n)$ |
| 5 | 5 | 0.7617 | $O(1.5762^n)$ |
| | | ... | |
| 10 | 10 | 0.9334 | $O(1.4496^n)$ |
| | | ... | |
| 20 | 20 | 0.9972 | $O(1.4156^n)$ |

Table 3: Some ratios and complexities for the algorithm `Bapprox`$^w$ on the $P2||\sum_j w_j U_j$ problem

**Remark 2.** *Notice that, if the same measure and conquer approach is applied to the unweighted case using the branching scheme of Figure 1 (correspondingly, the recursion $T(p) \leq T(p - k(1 - \delta)) + T(p - k)$ holds), the analysis provides as expected the same complexity illustrated in Table 1.*

## 7. Conclusions

In this paper we focused on the solution of some scheduling problems involving the minimization of the weighted and unweighted number of tardy jobs. We have shown that these $\mathcal{NP}$-hard problems cannot be approximated in polynomial time leading by the way to the design of exponential-time approximation algorithms. We also provide fixed parameter tractable algorithms which solve to optimality some of the tackled problems.

The proposed approximation algorithms relies on the combination of branching algorithms and the solution of decision problems. Some of them also embed a preprocessing step. Two points are remarkable: first, the approximation nature of these algorithms comes from the branching phase that does not explore the whole solution space. Second, it is expected that, to decrease the worst-case time complexity of the approximation algorithms, one should decrease the time complexity of solving the decision problems to be solved.

Exponential-time approximation algorithms enable to approximate hard problems that, under the assumption $\mathcal{P} \neq \mathcal{NP}$, cannot be approximated in polynomial time. This constitutes a very interesting approach in the field of scheduling which is worthy of further investigation.


## Acknowledgements

This work has been partially supported by "Ministero dell'Istruzione, dell'Università e della Ricerca" Award "TESUN-83486178370409 finanziamento dipartimenti di eccellenza CAP. 1694 TIT. 232 ART. 6".


## Appendix

We prove the results stated in Lemma 3. They are a direct consequence of the two lemmas below, applied with $\alpha = \frac{1}{c}$, knowing that $\lfloor \frac{n}{c} \rfloor = \frac{n}{c} + O(1)$.

First, we introduce the $\Theta^*$ notation. It is the equivalence associated with the $\mathcal{O}^*$ preorder. Given $A(n)$ and $B(n)$ two complexities, $A(n) = \mathcal{O}^*(B(n))$ stands for $A(n) = \mathcal{O}^*(B(n))$ and $B(n) = \mathcal{O}^*(A(n))$. $\Theta^*$ implies $\mathcal{O}^*$ with tight bound.

**Lemma 4.** *Given $0 < \alpha < 1$, let $\ell \leq n$ be an integer related to $n$ by $\ell = \alpha n + O(1)$. We have $\binom{n}{\ell} = \Theta^*\left(\left(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}\right)^n\right) = \Theta^*\left(2^{H(\alpha)n}\right)$*

Proof. For any logarithm, we have $A = \Theta^*(B)$ iff $\log A = \log B + O(\log n)$.
Consider Stirling's formula: $n! \sim (\frac{n}{e})^n \sqrt{2\pi n}$, with $e$ being Euler's number. Apply natural logarithm:

$$\ln n! = n \ln n - n + O(\ln n)$$

Thus, for any $\alpha$:

$$\ln(\alpha n + O(1))! \quad = \quad (\alpha n + O(1)) \ln((\alpha n)(1 + O(\frac{1}{n}))) + O(\ln(\alpha n + O(1)))$$

After development and simplification:

$$\ln(\alpha n + O(1))! = \alpha n \ln(\alpha n) - \alpha n + O(\ln n)$$

That is:

$$(\alpha n + O(1))! = \Theta^*\left(\left(\frac{\alpha n}{e}\right)^{\alpha n}\right)$$

Now, take the definition of $\binom{n}{\ell}$ and simplify:

$$\binom{n}{\ell} = \frac{n!}{\ell!(n-\ell)!} = \Theta^*\left(\frac{\left(\frac{n}{e}\right)^n}{\left(\frac{\alpha n}{e}\right)^{\alpha n}\left(\frac{(1-\alpha)n}{e}\right)^{(1-\alpha)n}}\right) = \Theta^*\left(\frac{1}{\alpha^{\alpha n}(1-\alpha)^{(1-\alpha)n}}\right)$$

And $\dfrac{1}{\alpha^{\alpha n}(1-\alpha)^{(1-\alpha)n}} = 2^{H(\alpha)}$ □

**Lemma 5.** *Let $\ell$ be a summation limit related to $n$ by $\ell = \alpha n + O(1)$, assuming $\alpha \leq \frac{1}{2}$. We have $\sum_{i=0}^{\ell}\binom{n}{i} = \Theta^*\left(2^{H(\alpha)n}\right)$*

Proof. If $\alpha < \frac{1}{2}$ then, for $n$ large enough, $\ell \leq \frac{n}{2}$ and $\binom{n}{0} \leq \binom{n}{1} \leq \ldots \leq \binom{n}{\ell}$. So:

$$\binom{n}{\ell} = \Theta^*\left(2^{H(\alpha)n}\right) \leq \sum_{i=0}^{\ell}\binom{n}{i} \leq (\ell+1)\binom{n}{\ell} = \Theta^*\left(2^{H(\alpha)n}\right)$$

If $\alpha = \frac{1}{2}$ then $H(\alpha) = H(\frac{1}{2}) = 1$ and $\binom{n}{i} \leq \binom{n}{\lfloor \frac{n}{2} \rfloor}$, $1 \leq i \leq \frac{n}{2}$. So:

$$\binom{n}{\ell} = \Theta^*\left(2^{H(\frac{1}{2})n}\right) \leq \sum_{i=0}^{\ell}\binom{n}{i} \leq (\ell+1)\binom{n}{\lfloor \frac{n}{2} \rfloor} = \Theta^*\left(2^{H(\frac{1}{2})n}\right) \quad \square$$


Bessy, S., Giroudeau, R., 2020. Parametrized complexity of a coupled-task scheduling problem. Journal of Scheduling 22, 305–313. doi:10.1007/s10951-018-0581-1.

Boardman, M., 2004. The egg-drop numbers. Mathematics Magazine 77, 368–372. doi:10.2307/3219201.

Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J., 2011. Scheduling partially ordered jobs faster than $2^n$, in: C. Demetrescu and M.M. Halldorsson (Eds): Proceedings of 19th Annual European Symposium (ESA 2011), Lecture Notes in Computer Science, pp. 299–310. doi:10.1007/978-3-642-23719-5_26.

Downey, R., Fellows, M., 1999. Parametrized complexity. Springer.

Escoffier, B., Paschos, V., Tourniaire, E., 2016. Super-polynomial approximation branching algorithms. RAIRO-Operations Research 50, 979–994. doi:10.1051/ro/2015060.

Fomin, F., Grandoni, F., Kratsch, D., 2009. A measure & conquer approach for the analysis of exact algorithms. Journal of the ACM 56, 1–32. doi:10.1145/1552285.1552286.

Garey, M., Johnson, D., 1979. Computers and Intractability: A guide to the theory of NP-completeness. Freeman, San Francisco (USA).

Garraffa, M., Shang, L., Della Croce, F., T'kindt, V., 2018. An exact exponential branch-and-merge algorithm for the single machine total tardiness problem. Theoretical Computer Science 745, 133–149. doi:10.1016/j.tcs.2018.05.040.

Gawiejnowicz, S., Kurc, W., 2020. New results for an open time-dependent scheduling problem. Journal of Scheduling, on line. doi:10.1007/s10951-020-00662-7.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5, 287–326. doi:10.1016/S0167-5060(08)70356-X.

Hall, L., 1998. Approximability of flow shop scheduling. Mathematical Programming, series B 82, 175–190. doi:10.1007/BF01585870.

Hermelin, D., Karhi, S., Pinedo, M., Shabtay, D., 2018. New algorithms for minimizing the weighted number of tardy jobs on a single machine. Annals of Operations Research, on line. doi:10.3934/jimo.2016.12.1465.

Jansen, K., Land, F., Land, K., 2013. Bounding the Running Time of Algorithms for Scheduling and Packing Problems, in: Dehne, F., Solis-Oba, R., Sack, J.R. (Eds.), Algorithms and Data Structures. Springer Berlin Heidelberg. volume 8037 of *Lecture Notes in Computer Science*, pp. 439–450. doi:10.1007/978-3-642-40104-6_38.

Jansen, K., Solis-Oba, R., Sviridenko, M., 2003. Makespan minimization in job shops: a linear time approximation scheme. SIAM Journal of Discrete Mathematics 16, 288–300. doi:10.1137/S0895480199363908.

Lawler, E.L., Moore, J.M., 1969. A functional equation and its application to resource allocation and sequencing problems. Management Science 16, 77–84. doi:10.1287/mnsc.16.1.77.

Lente, C., Liedloff, M., Soukhal, A., T'kindt, V., 2013. On an extension of the sort & search method with application to scheduling theory. Theoretical Computer Science 511, 13–22. doi:10.1016/j.tcs.2013.05.023.

Lenté, C., Liedloff, M., Soukhal, A., T'Kindt, V., 2014. Exponential Algorithms for Scheduling Problems. Technical Report. University of Tours. URL: https://hal.archives-ouvertes.fr/hal-00944382.

Leung, J.T., Yu, V., 1994. Heuristic for minimizing the number of late jobs on two processors. International Journal of Foundations of Computer Science 5, 261–279. doi:10.1142/S0129054194000141.

Lin, Y., Wang, X., 2007. Necessary and sufficient conditions of optimality for some classical scheduling problems. European Journal of Operational Research 176, 809–818. doi:10.1016/j.ejor.2005.09.017.

Marx, D., 2008. Parametrized complexity and approximation algorithms. The Computer Journal 51, 60–78. doi:10.1093/comjnl/bxm048.

Mnich, M., van Bevern, R., 2018. Parametrized complexity of machine scheduling: 15 open problems. Computers & Operations Research 100, 254–261. doi:10.1016/j.cor.2018.07.020.

Mnich, M., Wiese, A., 2015. Scheduling and fixed-parameter tractability. Mathematical Programming B 154, 85–103. doi:10.1007/s10107-014-0830-9.

Paschos, V., 2015. When polynomial approximation meets exact computation. 4'OR 13, 227–245. doi:10.1007/s10288-015-0294-7.

Pinedo, M., 2016. Scheduling - Theory, Algorithms, and Systems. Springer.

Robson, J., 1986. Algorithms for maximum independent sets. Journal of Algorithms 7, 425–440. doi:10.1016/0196-6774(86)90032-5.

Sevastianov, S., Woeginger, G., 1998. Makespan minimization in open shops: a polynomial time approximation scheme. Mathematical Programming, series B 82, 191–198. doi:10.1007/BF01585871.

Shang, L., Lenté, C., Liedloff, M., T'Kindt, V., 2018. Exact exponential algorithms for 3-machine flowshop scheduling problems. Journal of Scheduling 21, 227–233. doi:10.1007/s10951-017-0524-2.

Woeginger, G., 2003. Exact algorithms for NP-hard problems: A survey. Lecture notes in computer science 2570, 185–207. doi:10.1007/3-540-36478-1_17.