Exponential time algorithms for just-in-time scheduling problems with common due date and symmetric weights

Vincent t'Kindt, Lei Shang, Federico Della croce

Noname manuscript No.
(will be inserted by the editor)

Exponential time algorithms for just-in-time scheduling problems with common due date and symmetric weights

Vincent T'kindt · Lei Shang · Federico Della Croce

Received: date / Accepted: date

Abstract This paper focuses on the solution, by exact exponential-time algorithms, of just-in-time scheduling problems when jobs have symmetric earliness/tardiness weights and share a non restrictive common due date. For the single machine problem, a *Sort & Search* algorithm is proposed with worst-case time and space complexities in $\mathcal{O}^*(1.4143^n)$. This algorithm relies on an original modeling of the problem. For the case of parallel machines, an algorithm integrating a *dynamic programming algorithm across subsets and machines* and the above *Sort & Search* algorithm is proposed with worst-case time and space complexities in $\mathcal{O}^*(3^n)$. To the best of our knowledge, these are the first worst-case complexity results obtained for non regular criteria in scheduling.

Keywords Single machine · Parallel machines · Just-in-time · Exponential time algorithms

1 Introduction

In this paper we revisit two well-known just-in-time scheduling problems under the light of exponential-time algorithms. We are given a set of n jobs, each job i being defined by a processing time p_i and a weight w_i, the latter reflecting the penalty induced by scheduling it early or tardy. All jobs share the same, non restrictive, common due date $d \geq \sum_i p_i$. The objective is to

Corresponding author: V. T'kindt.
V. T'kindt and L. Shang
University of Tours, Laboratory of Fundamental and Applied Computer Science (EA 6300), ERL CNRS 6305, 64 avenue Jean Portalis, 37200 Tours, France
E-mail: {tkindt,lei.shang}@univ-tours.fr. Tel: +33247361427.

F. Della Croce
Politecnico di Torino, DIGEP, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
CNR, IEIIT, Torino, Italy,
E-mail: federico.dellacroce@polito.it

find a schedule s of jobs such that $\sum_i w_i(E_i(s) + T_i(s))$ is minimized, with $T_i(s) = \max(C_i(s) - d; 0)$ and $E_i(s) = \max(d - C_i(s); 0)$. Notice that the mention of schedule s may be omitted whenever there is no ambiguity. In the first tackled problem a single machine is assumed to be available to process the jobs and, following the standard three-fields notation, this problem can be referred to as $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$. The second tackled problem is the extension to the case where several identical parallel machines are available and the corresponding problem is referred to as $P|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$.

Just-in-Time scheduling problems have been the matter of numerous published works and comprehensive surveys have been provided by T'kindt and Billaut (2006) and Jozefowska (2007), among others. The single machine problem with a non restrictive common due date and unit weights has been shown to be polynomially solvable by Kanet (1981). However, for the problem with symmetric weights considered in this paper, the problem turns out to be $\mathcal{NP}$-hard as shown by Hall and Posner (1991). Interestingly, several remarkable properties, summarized in Property 1, have been established along the years on problem $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$. They notably induce that the hardness of the problem comes from deciding for each job if it is better to schedule it early or tardy.

Property 1 (Baker and Scudder (1989)) There exist optimal solutions to the $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ problem satisfying the following properties:

1. there are no machine idle times between two consecutive jobs,
2. the first scheduled job can start at a time greater than 0,
3. there exists a job which exactly completes at time d,
4. the class of V-shape schedules is dominant.

A schedule is said to be V-shaped, for the symmetric weight problem, iff all early jobs are sequenced by decreasing value of the ratio $\frac{p_i}{w_i}$ (WLPT rule) while all tardy jobs are sequenced by increasing value of the ratio $\frac{p_i}{w_i}$ (WSPT rule). Figure 1 presents a schedule satisfying Property 1.

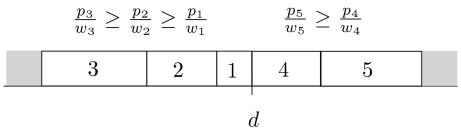


Fig. 1 A 5-job schedule satisfying Property 1

Several exact algorithms exist for a generalization of the problems tackled in this paper considering arbitrary weights that can be well applied to the case of symmetric weigths. These algorithms, however, focus on the solution of randomly generated instances with the purpose of being efficient in practice. The problem of scheduling identical parallel machines, denoted by $P|d_i = d \geq$

$\sum_i p_i | \sum_i w_i(E_i + T_i)$, is also $\mathcal{NP}$-hard. Again, there exist exact algorithms to solve the more general problem with arbitrary weights but the purpose to be effective on the average on randomly generated intances.

Purpose of this paper is to revisit these two problems in the light of *exact exponential-time algorithms* as presented in the seminal book of Fomin and Kratsch (2010). These algorithms are designed to solve NP-hard problems with the intent to reach the lowest possible worst-case time complexity. When dealing with exponential algorithms, we usually make use of the $\mathcal{O}^*(\cdot)$ notation: let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on the instance size (usually the number of jobs n for scheduling). Then, we express running-time bounds of the form $\mathcal{O}(p(n) \cdot T(n)))$ as $\mathcal{O}^*(T(n))$.
Since the last decades, much effort has been put in the literature into the solution of graph problems as surveyed by Woeginger (2003, 2004) and Fomin and Kratsch (2010). Quite recently, a few works on scheduling problems have been published. For an introduction to exponential-time algorithms for scheduling we refer to Lenté et al. (2014). Following the early work of Held and Karp (1962), Woeginger (2003) who proposed a $O^*(2^n)$ time and space *dynamic programming across the subsets* algorithm to solve the $1|prec|\sum w_iC_i$ problem. Notably, this algorithm is also valid for a set of scheduling problems whose optimal solution can be reduced to the recursive solution of subproblems (see Fomin and Kratsch (2010)). This leads to $O^*(2^n)$ time and space dynamic programming algorithms for the $1|\tilde{d}_i|\sum w_iC_i$ and the $1|d_i|\sum w_iT_i$ problems, and to an $O^*(3^n)$ time and space algorithm for the $1|r_i, prec|\sum C_i$ problem. Later on, Cygan et al. (2011) provided an $O^*((2-10^{-10})^n)$ time algorithm for the $1|prec|\sum C_i$ problem. For the $1|d_i|\sum w_iU_i$ problem, Lenté et al. (2014) proposed a $O^*(1.4143^n)$ *Sort & Search* algorithm. For the $1|d_i|\sum T_i$ problem an algorithm with $O^*((2+\epsilon)^n)$ time complexity but polynomial space complexity has been proposed by Shang et al. (2018). This algorithm relies on the exploration of a search tree with a dedicated node merging procedure. An interesting generic approach for designing exponential-time algorithms has been proposed by Lente et al. (2013) who provide a generalization of the well-known *Sort & Search* algorithm to a general class of problems called *multiple constraints problems*. This has induced several *Sort & Search* algorithms having time and space complexities in $O^*(1.4143^n)$ for the $P2||C_{max}$ and $F2||C^k_{max}$ problems, in $O^*(1.7320^n)$ for the $P2|d_i|\sum w_iU_i$ and $P3||C_{max}$ problems, in $O^*((m+1)^{\frac{n}{2}}(\frac{n}{2})^{m+2})$ for the $P|d_i|\sum w_iU_i$ problem and in $O^*(m^{\frac{n}{2}}(\frac{n}{2})^{m+1})$ for the $P||C_{max}$ problem. We also point out the work of Shang et al. (2017) who proposed a so-called *Pareto dynamic programming* and applied it to a set of flowshop scheduling problems. They notably showed that the $F3||C_{max}$ problem can be solved in $O^*(3^n)$ time and space and that two more general problems referred to as $F3||\max_i(f_i)$ and $F3||\sum_i f_i$, with the f_i's being increasing functions of job completion times, can be solved in $O^*(5^n)$ time and space. Also based on *dynamic programming*, Gromicho et al. (2012) solved the job shop scheduling problem in $O(p^{2n}_{max}(m+1)^n)$ time, with $p_{max} = \max_{i=1..n}(p_i)$. Jansen et al. (2013b) developed a framework that is able to solve a number

of scheduling and packing problems in $2^{O(n)}$ time. Finally, it is interesting to notice that Jansen et al. (2013a) provided some results, under the *Exponential Time Hypothesis*, on the existence of lower bounds on the worst-case time complexities of scheduling problems.

To the best of our knowledge, no exponential-time algorithm has been proposed for scheduling problems involving the minimization of non increasing functions of job completion times, like the $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i+T_i)$ and $P|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ problems tackled in this paper. In section 2 we provide a *Sort & Search* algorithm running in $O^*(1.4143^n)$ time and space for the single machine problem. The generalization to the identical parallel machines setting is considered in section 3 and a $O^*(3^n)$ time algorithm is given. Section 4 concludes this paper and presents potential future research directions.

2 Single machine scheduling

Among the known techniques to derive exponential-time algorithms (see, *e.g.*, Fomin and Kratsch (2010)), there is *Sort & Search* initially proposed by Horowitz and Sahni (1974) to solve the knapsack problem in $O^*(1.4143^n)$ time and space and later on extended by Lente et al. (2013) who also applied it to a set of scheduling problems. Roughly speaking, it consists in separating an input instance into two equal-size instances, then in enumerating all partial solutions for each sub-instance and finally find the optimal solution of the input instance by recombining in a suitable way all those partial solutions taking each time one from each sub-instance. This "complexity breaking" is done at the detriment of the space complexity which turns out to be exponential. Before presenting the details of the proposed algorithm, we highlight the fact that the $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ problem can be solved in $O^*(2^n)$ time and space by a brute-force algorithm that enumerates all possible assignments of jobs to the sets of early and tardy jobs.

Without loss of generality, assume that n is even and that jobs are indexed such that $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_n}{w_n}$. In the remainder we implicitly make use of the results in Property 1 to elaborate our *Sort & Search* algorithm. For any given instance I of n jobs, let $I_1 = \{1, \ldots, \frac{n}{2}\}$ and $I_2 = \{\frac{n}{2}+1, \ldots, n\}$ be a decomposition into two equal-size sub-instances. By enumeration, done in $\mathcal{O}^*(2^{\frac{n}{2}})$ time, we can build set $\mathcal{S}_1 = \{s_j^1 | j = 1, \ldots, 2^{|I_1|}\}$ (resp. $\mathcal{S}_2 = \{s_k^2 | k = 1, \ldots, 2^{|I_2|}\}$) which is the set of all possible solutions built from sub-instance I_1 (resp. I_2). We have $|\mathcal{S}_1| = |\mathcal{S}_2| = 2^{\frac{n}{2}}$. Figure 3 shows, for an instance I, a complete schedule $s = s_j^1 // s_k^2$ decomposed into two partial solutions $s_j^1 = \{\epsilon_j^1; \tau_j^1\} \in \mathcal{S}_1$ and $s_k^2 = \{\epsilon_k^2; \tau_k^2\} \in \mathcal{S}_2$, with ϵ_x^y (resp. τ_x^y) referring to a schedule of early jobs (resp. tardy jobs). Besides, t_j^s refers to the starting time of the first job in ϵ_j^1 or equivalently to the completion time of the last job in ϵ_k^2. Symmetrically, t_j^c refers to the completion time of the last job in τ_j^1 or equivalently to the

starting time of the first job in τ_k^2.

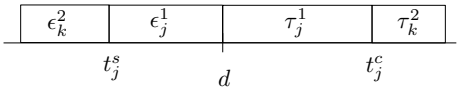


Fig. 2 Decomposition of a complete schedule into two sub-schedules s_j^1 and s_k^2

From this decomposition scheme, it follows that solving the $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i+T_i)$ problem reduces to finding the best partial schedules $s_j^1 \in \mathcal{S}_1$ and $s_k^2 \in \mathcal{S}_2$ such that schedule $s = s_j^1 // s_k^2$ has the minimum $\sum_i w_i(E_i+T_i)$ value. In the remainder, we show that finding such a schedule s can be done in the worst case with a time complexity not worse than that of building sets $\mathcal{S}_1$ and $\mathcal{S}_2$. First, we can show the following result.

Proposition 1 *Let us denote by $s = s_j^1 // s_k^2$ a complete schedule, and by $f_{jk} = \sum_{i \in s} w_i(E_i(s) + T_i(s))$ its objective function value. We have:*

$$f_{jk} = f_j + c_k + a_k t_j^s,$$

with $a_k = \sum_{i \in \tau_k^2} w_i - \sum_{i \in \epsilon_k^2} w_i$, $f_j = \sum_{i \in s_j^1} w_i(E_i(s_j^1) + T_i(s_j^1))$ and $c_k = \sum_{i \in s_k^2} w_i(E_i(s_k^2)+T_i(s_k^2))+d(\sum_{i \in \epsilon_k^2} w_i - \sum_{i \in \tau_k^2} w_i)+\sum_{i \in \tau_k^2} w_i \sum_{i \in I_1} p_i$. Notice that f_j and t_j^s depend only on s_j^1, while c_k and a_k depend only on s_k^2.

Proof Consider Figure 3 and the decomposition of a schedule $s = s_j^1 // s_k^2$. We have:

$$\begin{aligned} f_{jk} &= \sum_{i \in s_j^1 // s_k^2} w_i(E_i(s_j^1 // s_k^2) + T_i(s_j^1 // s_k^2)) \\ &= \sum_{i \in s_j^1} w_i(E_i(s_j^1) + T_i(s_j^1)) + \sum_{i \in \epsilon_k^2} w_i E_i(\epsilon_k^2) + (d - t_j^s) \sum_{i \in \epsilon_k^2} w_i \\ &\quad + \sum_{i \in \tau_k^2} w_i T_i(\tau_k^2) + (t_j^c - d) \sum_{i \in \tau_k^2} w_i \end{aligned}$$

with $E_i(\epsilon_k^2)$ (resp. $T_i(\tau_k^2)$) the earliness of job $i \in \epsilon_k^2$ (resp. job $i \in \tau_k^2$) when partial schedule s_k^2 is scheduled around the common due date d (partial schedule s_j^1 is then omitted). As $t_j^c = t_j^s + \sum_{i \in I_1} p_i$, we can rewrite:

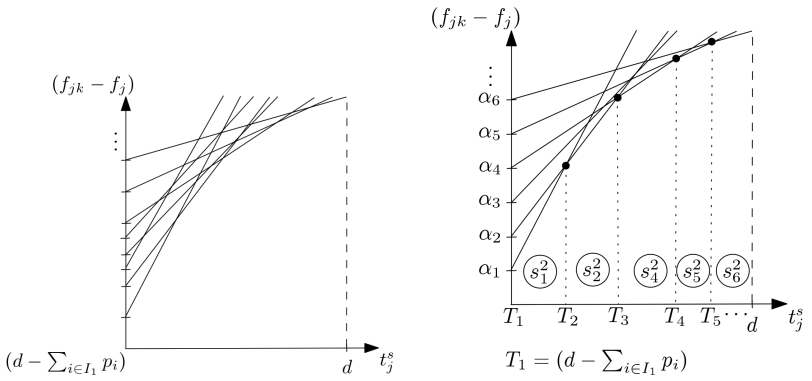
$$\begin{aligned} f_{jk} &= \sum_{i \in s_j^1} w_i(E_i(s_j^1) + T_i(s_j^1)) + \sum_{i \in s_k^2} w_i(E_i(s_k^2) + T_i(s_k^2)) \\ &\quad + d(\sum_{i \in \epsilon_k^2} w_i - \sum_{i \in \tau_k^2} w_i) + \sum_{i \in \tau_k^2} w_i \sum_{i \in I_1} p_i \\ &\quad + t_j^s(\sum_{i \in \tau_k^2} w_i - \sum_{i \in \epsilon_k^2} w_i) \\ &= f_j + c_k + a_k t_j^s \end{aligned}$$

□

From Proposition 1 it follows that for any given partial schedule s_j^1, f_j and t_j^s can be computed in $\mathcal{O}(n)$ time. This is also the case for c_k and a_k whenever partial schedule s_k^2 is given. In the remainder we assume that these

values are computed when building sets $\mathcal{S}_1$ and $\mathcal{S}_2$, thing which does not affect the $\mathcal{O}^*(2^{\frac{n}{2}})$ time complexity required by the *Sort & Search* algorithm to build these sets. For a given partial schedule $s_j^1 \in \mathcal{S}_1$ the algorithm needs to find a schedule $s_k^2 \in \mathcal{S}_2$ such that f_{jk} is minimum. The optimal solution associated to instance I is then given as the best complete solution obtained, starting from $j = 1$ to $j = |\mathcal{S}_1|$.

Now, let us turn to the search of the best schedule s_k^2 when s_j^1 is fixed. We separate set $\mathcal{S}_2$ into sub-sets $\mathcal{S}_2^+ = \{s_k^2 \in \mathcal{S}_2 | a_k \geq 0\}$ and $\mathcal{S}_2^- = \{s_k^2 \in \mathcal{S}_2 | a_k < 0\}$ and the search for the best partial schedule s_k^2 is done first in $\mathcal{S}_2^+$ and next in $\mathcal{S}_2^-$. We show that the search in $\mathcal{S}_2^+$ can be done in $\mathcal{O}^*(2^{\frac{n}{2}})$ time knowing that the same result holds for searching in $\mathcal{S}_2^-$. First, remark that whatever is the fixed partial schedule $s_j^1 \in \mathcal{S}_1$, function f_{jk} is minimized when $c_k + a_k t_j^s = (f_{jk} - f_j)$ is minimum. So, finding the best s_k^2 complementing a given s_j^1 is equivalent to identifying the minimal function $(f_{jk} - f_j)$ for the value t_j^s associated to s_j^1 . Figure 3a gives an example of $(f_{jk} - f_j)$ functions for all values of $t_j^s \in [d - \sum_{i \in I_1} p_i; d]$.



(a) Evolution of $(f_{jk} - f_j)$ over time (b) Intervals identification

Fig. 3 Contributions of partial schedules $s_k^2 \in \mathcal{S}_2^+$

We also re-index partial sequences $s_k^2 \in \mathcal{S}_2^+$ by increasing values of $\alpha_k = (c_k + a_k(d - \sum_{i \in I_1} p_i)) = \sum_{i \in s_k^2} w_i(E_i(s_k^2) + T_i(s_k^2)) + \sum_{i \in \epsilon_k^2} w_i \sum_{i \in I_1} p_i$. Then, we remove all s_k^2 such that $\alpha_k \geq \alpha_{k-1}$ and $a_k \geq a_{k-1}$, since in that case $c_k + a_k t_j^s \geq c_{k-1} + a_{k-1} t_j^s$ holds for any t_j^s value. This preprocessing step implies that functions $(f_{kj} - f_j)$ and $(f_{k-1,j} - f_j)$ intersect. Figure 3b shows an update of Figure 3a after the preprocessing on $\mathcal{S}_2^+$. By means of Algorithm 1 it is possible to compute couples $(T_\ell, s_{\pi(\ell)}^2)$ with the meaning that whenever $t_j^s \in [T_\ell; T_{\ell+1}[$, partial schedule $s_{\pi(\ell)}^2$ leads to the complete schedule $s = s_j^1 // s_{\pi(\ell)}^2$ with minimum cost. Here, $\pi(\ell)$ is the number of the sequence s_k^2 associated to the ℓ-th time interval $[T_\ell; T_{\ell+1}[$.

Algorithm 1 Computation of time intervals on $\mathcal{S}_2^+$

1: $T_1 = (d - \sum_{i \in I_1} p_i)$, $T_2 = d$;
2: $\mathcal{A}^+ = \{(T_1; s_1^2), (T_2; \emptyset)\}$;
3: **for** $k = 2..|\mathcal{S}_2^+|$ **do**
4: Let $(T_{|\mathcal{A}^+|-1}; s^2_{\pi(|\mathcal{A}^+|-1)})$ be the couple in position $(|\mathcal{A}^+| - 1)$ in $\mathcal{A}^+$;
5: **if** $(c_k + a_k d < c_{\pi(|\mathcal{A}^+|-1)} + a_{\pi(|\mathcal{A}^+|-1)} d)$ **then**
6: Apply a dichotomic search over $\mathcal{A}^+$ to identify $(T_\ell; s^2_{\pi(\ell)})$ such that:
7: $\left(c_{\pi(\ell)} + a_{\pi(\ell)} T_{\ell+1} \geq c_k + a_k T_{\ell+1}\right)$ and $\left(c_k + a_k T_\ell \geq c_{\pi(\ell)} + a_{\pi(\ell)} T_\ell\right)$;
8: Remove from $\mathcal{A}^+$ all couples $(T_p; s^2_{\pi(p)})$, $\forall p = \ell + 1, ..., |\mathcal{A}^+|$;
9: Add in $\mathcal{A}^+$ (at the end) couples $(T_{\ell+1} = \frac{c_\ell - c_k}{a_k - a_\ell}; s_k^2)$ and $(T_{\ell+2} = d; \emptyset)$;
10: **end if**
11: **end for**
12: Return $\mathcal{A}$;

Figure 4 illustrates how Algorithm 1 works. The initialization of set $\mathcal{A}^+$ consists in determining couples $(T_1 = (d - \sum_{i \in I_1} p_i); s_1^2)$ and $(T_2 = d; \emptyset)$. Next, each partial schedule s_k^2 is iteratively considered in order to update set $\mathcal{A}^+$ (lines 3-11). Two situations may occur (lines 4-5):

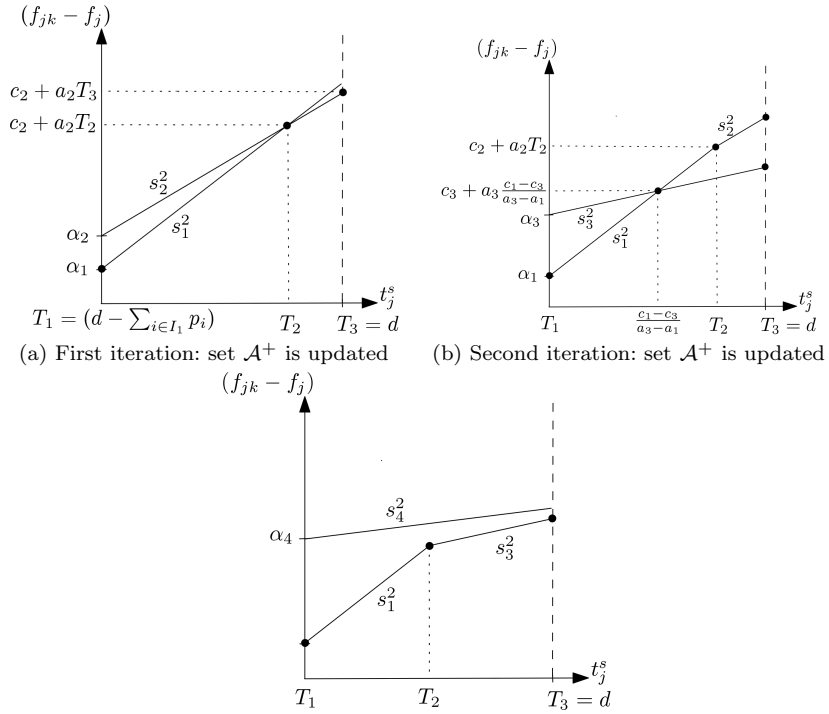
1. either function $(f_{jk} - f_j)$ intersects in $[d - \sum_{i \in I_1} p_i; d]$ some function $(f_{j\pi(\ell)} - f_j)$ corresponding to partial schedule $s^2_{\pi(\ell)} \in \mathcal{A}^+$,
2. or it does not intersect in $[d - \sum_{i \in I_1} p_i; d]$ any function.

In the first case, by a dichotomic search over $\mathcal{A}^+$, the time interval $[T_\ell; T_{\ell+1}]$ on which $(f_{jk} - f_j)$ intersects $(f_{j\pi(\ell)} - f_j)$ is determined (lines 6-7). Then, the update of set $\mathcal{A}$ is done by removing some couples and add two new ones (Figures 4a and 4b). In the second case (Figure 4c) no modifications are done on $\mathcal{A}$.

Proposition 2 *Algorithm 1 runs in the worst-case in* $\mathcal{O}^*(2^{\frac{n}{2}})$ *time and returns at most* $(2^{\frac{n}{2}} + 1)$ *couples* $(T_\ell; s^2_{\pi(\ell)})$.

Proof The worst-case time complexity is given by the number of loops in Algorithm 1 multiplied by the complexity of doing one loop. First, notice that lines 8-9 can be done in constant time by making use of appropriate data structures. The dichotomic search in lines 6-7 is done at most in $\mathcal{O}(\log(|\mathcal{S}_2^+|))$ time. As there are at most $2^{\frac{n}{2}}$ partial schedules s_k^2 in $\mathcal{S}_2^+$, the total running time of the algorithm is in $\mathcal{O}^*(|\mathcal{S}_2^+| \log(|\mathcal{S}_2^+|))$=$\mathcal{O}^*(2^{\frac{n}{2}})$.
The cardinality of set $\mathcal{A}^+$ is at most $(2^{\frac{n}{2}}+1)$ which is achieved when all partial schedules in $\mathcal{S}_2^+$ lead to an update of $\mathcal{A}^+$ (lines 6-9) and when the dichotomic search systematically returns $\ell = (|\mathcal{A}^+| - 1)$. □

For any given $s_j^1 \in \mathcal{S}_1$, searching the best complement $s_k^2 \in \mathcal{S}_2^+$ is then equivalent to searching in set $\mathcal{A}^+$ which is sorted by increasing values of T_ℓ: this relates to the search of the value T_ℓ such that $T_\ell \leq t_j^s \leq T_{\ell+1}$, and then $s^2_{\pi(\ell)}$ is the searched solution. Noteworthy, the search in $\mathcal{S}_2^-$ requires to compute the set $\mathcal{A}^-$ in a way similar to Algorithm 1. The global *Sort & Search*



(a) First iteration: set $\mathcal{A}^+$ is updated (b) Second iteration: set $\mathcal{A}^+$ is updated

(c) Third iteration: set $\mathcal{A}^+$ is not updated

Fig. 4 Illustration of Algorithm 1

algorithm for solving the $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ problem is given in Algorithm 2.

Theorem 1 *Algorithm 2 solves the* $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ *problem in* $\mathcal{O}^*(2^{\frac{n}{2}})$ *time and space in the worst case.*

Proof As each set I_1 and I_2 has $\frac{n}{2}$ jobs, the building of sets $\mathcal{S}_1$, $\mathcal{S}_2^+$ and $\mathcal{S}_2^-$ requires $\mathcal{O}^*(2^{\frac{n}{2}})$ time by a brute force enumeration algorithm. Notice that, for each partial schedule $s_j^1 \in \mathcal{S}_1$, we compute in polynomial time t_j^s and $f_j = \sum_{i \in s_j^1} w_i(E_i(s_j^1) + T_i(s_j^1))$. Besides, for each $s_k^2 \in \mathcal{S}_2^+ \cup \mathcal{S}_2^-$ we compute in polynomial time $a_k = \sum_{i \in \tau_k^2} w_i - \sum_{i \in \epsilon_k^2} w_i$ and $c_k = \sum_{i \in s_k^2} w_i(E_i(s_k^2) + T_i(s_k^2)) + d(\sum_{i \in \epsilon_k^2} w_i - \sum_{i \in \tau_k^2} w_i) + \sum_{i \in \tau_k^2} w_i \sum_{i \in I_1} p_i$. This also implies that the space required for storing these three sets is in $\mathcal{O}^*(2^{\frac{n}{2}})$. Computation of sets $\mathcal{A}^+$ and $\mathcal{A}^-$ can also be done in $\mathcal{O}^*(2^{\frac{n}{2}})$ time and space.
Let us turn to the search phase of the algorithm. For each $s_j^1 \in \mathcal{S}_1$ we perform two dichotomic searches requiring each at most $\mathcal{O}(\log(2^{\frac{n}{2}})) = \mathcal{O}(n)$ time. As we have $|\mathcal{S}_1| \leq 2^{\frac{n}{2}}$, the search phase requires at most $\mathcal{O}^*(2^{\frac{n}{2}})$ time.□

Algorithm 2 Solution of the $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ problem

{**Sort phase**}
Let $I_1 = \{1, \ldots, \frac{n}{2}\}$ and $I_2 = \{\frac{n}{2} + 1, \ldots, n\}$;
Compute sets $\mathcal{S}_1$, $\mathcal{S}_2^+$ and $\mathcal{S}_2^-$;
Compute set $\mathcal{A}^+$ (Algorithm 1) and set $\mathcal{A}^-$;
{**Search phase**}
Let $s^* = \emptyset$ and $f^* = +\infty$;
for $s_j^1 \in \mathcal{S}_1$ **do**
  Apply a dichotomic search over $\mathcal{A}^+$ to identify $(T_\ell; s^2_{\pi(\ell)})$ such that:
    $T_\ell \leq t_j^s \leq T_{\ell+1}$;
  if $(\sum_i w_i(E_i(s_j^1 // s^2_{\pi(\ell)}) + T_i(s_j^1 // s^2_{\pi(\ell)}) < f^*)$ **then**
    $s^* = s_j^1 // s^2_{\pi(\ell)}$ and $f^* = \sum_i w_i(E_i(s_j^1 // s^2_{\pi(\ell)}) + T_i(s_j^1 // s^2_{\pi(\ell)})$;
  end if
  Apply a dichotomic search over $\mathcal{A}^-$ to identify $(T_h; s^2_{\pi'(h)})$ such that:
    $T_h \leq t_j^s \leq T_{h+1}$;
  if $(\sum_i w_i(E_i(s_j^1 // s^2_{\pi'(h)}) + T_i(s_j^1 // s^2_{\pi'(h)}) < f^*)$ **then**
    $s^* = s_j^1 // s^2_{\pi'(h)}$ and $f^* = \sum_i w_i(E_i(s_j^1 // s^2_{\pi'(h)}) + T_i(s_j^1 // s^2_{\pi'(h)})$;
  end if
end for
Return s^* and f^*;

3 Identical parallel machines scheduling

We consider in this section the problem with a set of m identical parallel machines available to process the jobset. We propose an exponential-time algorithm using the one proposed in section 2 and exploit a machine decomposition scheme introduced by Lenté et al. (2014) for the minimization of regular costs functions. The latter is introduced hereafter as a *dynamic programming algorithm across subsets and machines*. For the sake of presentation, assume that there exists an integer value μ such that $m = 2^\mu$ (in case of $m \neq 2^\mu$ a similar dynamic program holds requiring, however, heavier and more tedious notation). Let (P_t), $1 \leq t \leq \mu = \log_2(m)$, be the scheduling problem obtained by considering only the first 2^t machines. As machines are identical, for any $t = 1, ..., \mu$, scheduling jobset $S \subseteq \{1, ..., n\}$ on the last 2^t machines is equivalent to schedule them on the first 2^t machines.

Let us focus on the dynamic programming scheme which enumerates across subsets and machines, and we denote by $Opt[t, S]$ the optimal solution value of problem (P_t) for jobset $S \subseteq \{1, ..., n\}$. We have the following recursive functions:

$$Opt[t, S] = \min_{S' \subseteq S}(Opt[t-1, S'] + Opt[t-1, S \backslash S']), \qquad \forall t = 1, ..., \mu, \ \forall S \subseteq \{1, ..., n\}, \quad (1)$$

$$Opt[t, \emptyset] = 0, \qquad \forall t = 0, ..., \mu. \quad (2)$$

The optimal solution of the $P|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ problem is obtained by computing $Opt[\mu, \{1, ..., n\}]$. Besides, notice that $Opt[0, S]$ relates

to the solution of the $1|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ with jobset S, which can be done by means of the *Sort & Search* algorithm introduced in section 2.

Algorithm 3 Solution of the $P|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ problem

1: **for** $S \subseteq \{1, ..., n\}$ **do**
2: Apply Algorithm 2 on set S and let s_1^* be the optimal solution obtained;
3: $Opt[0, S] = \sum_i w_i(E_i(s_1^*) + T_i(s_1^*))$;
4: **end for**
5: Compute $f^* = Opt[\mu, \{1, ..., n\}]$ by means of the recursive functions $(1, 2)$ and deduce the corresponding solution s^*;
6: Return s^* and f^*;

Theorem 2 *Algorithm 3 solves the* $P|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i+T_i)$ *problem in* $\mathcal{O}^*(3^n)$ *time and* $\mathcal{O}^*((1+\sqrt{2})^n)$ *space in the worst case.*

Proof The first part of Algorithm 3 consists in solving all possible single machine problems for any subset $S \subseteq \{1, ..., n\}$. Thus, the required time complexity is bounded by:

$$\sum_{k=0}^{n} \binom{n}{k} (\sqrt{2})^k = (1 + \sqrt{2})^n,$$

by Newton's binomial theorem. Then, the first part of Algorithm 3 runs in $\mathcal{O}^*((1 + \sqrt{2})^n)$ time. The space requirement is also in $\mathcal{O}^*((1 + \sqrt{2})^n)$.
The second part consists in computing recursively the value of $Opt[\mu, \{1, ..., n\}]$ through an enumeration of all subsets S on all problems (P_t). For a given value t, we count the number of problems $Opt[t - 1, S']$ that have to be solved. For a given k we have $\binom{n}{k}$ possible sets S of size k, and so $\binom{n}{k}2^k$ subsets $S' \subseteq S$. Then, for a given t value, the number of problems $Opt[t - 1, S']$ to consider is equal to:

$$\sum_{k=0}^{n} \binom{n}{k} 2^k = 3^n$$

by Newton's binomial theorem. As there are $\log_2(m)$ values of t, the worst-case time complexity of the dynamic programming phase is in $\mathcal{O}^*(3^n)$ time. As we need to store in memory the values of $Opt[t, S]$, the space requirement is in $\mathcal{O}^*(2^n)$.□

To conclude this section, notice that Algorithm 3 can be slightly adapted to solve the problem with uniform parallel machines and unrelated parallel machines. Consider the more general problem with unrelated parallel machines, then $p_{i,j}$ refers to the processing time of job i when processed by machine j. The dynamic programming phase is still valid but needs to be rewritten as follows:

$$Opt[t_{min}, t_{\max}, S] =$$
$$\min_{S' \subseteq S}(Opt[t_{min}, \lfloor \tfrac{t_{min}+t_{max}}{2} \rfloor, S'] + Opt[\lceil \tfrac{t_{min}+t_{max}}{2} \rceil, t_{max}, S \backslash S']),$$
$$\forall 1 \leq t_{min} \leq t_{max} \leq \mu, \forall S \subseteq \{1, ..., n\},$$
$$Opt[t_{min}, t_{\max}, \emptyset] = 0, \forall 0 \leq t_{min} \leq t_{max} \leq \mu,$$

with $Opt[t_{min}, t_{\max}, S]$ the problem of scheduling jobset S on the set of machines $\{2^{t_{min}}; ...; 2^{t_{\max}}\}$. Then, the single machine problems appear whenever $t_{min} = t_{max}$. The first phase of Algorithm 3 has to be adjusted by applying the *Sort & Search* algorithm on instances with processing times corresponding to the considered machine. Correspondingly, the following corollary holds.

Corollary 1 *The* $R|d_i = d \geq \sum_i p_i| \sum_i w_i(E_i + T_i)$ *problem can be solved in* $\mathcal{O}^*(3^n)$ *time and* $\mathcal{O}^*((1+\sqrt{2})^n)$ *space in the worst case.*

4 Conclusions and future directions

This paper focuses on the solution, by exact exponential-time algorithms, of just-in-time scheduling problems when jobs have symmetric earliness/tardiness weights and share a non restrictive common due date. For the single machine problem, a *Sort & Search* algorithm is proposed with worst-case time complexity in $\mathcal{O}^*(1.4143^n)$. For the case of parallel machines, an algorithm integrating a *dynamic programming algorithm across subsets and machines* and the above *Sort & Search* algorithm is proposed with worst case time and space complexities in $\mathcal{O}^*(3^n)$. To the best of our knowledge, these are the first worst-case complexity results obtained for non regular criteria in scheduling. Usually, minimizing such criteria implies the solution of an optimal timing problem (calculation of optimal job starting times) in addition to the problem of assigning and sequencing jobs to machines. However, we show that due to the structure induced by the common due date and the presence of symmetric weights, the optimal timing problem can be taken into account in the *Sort & Search* approach by building data structures integrating a temporal dimension. This is unconventional with respect to the classic *Sort & Search* approach of Horowitz and Sahni (1974).
Just-in-time scheduling problems seem to be very challenging problems, in the context of exact exponential-time algorithms. The presence of optimal timing problems creates an extra complexity which is not easy to take into account. It could be then very interesting to investigate the existence of worst-case time and space complexity results for problems with arbitrary weights and/or due dates.

Acknowledgement 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".

References

K.R. Baker and G.D. Scudder. On the assignment of optimal due dates. *Journal of the Operational Research Society*, 40:93–95, 1989.

M. Cygan, Ma. Pilipczuk, Mi. Pilipczuk, and J.O. Wojtaszczyk. 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*, volume 6942, pages 299–310, 2011.

F. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Springer, 2010.

J.A.S. Gromicho, J.J. Van Hoorn, F. Saldanha da Gama, and G.T. Timmer. Solving the job-shop scheduling problem optimally by Dynamic Programming. *Computers & Operations Research*, 39(12):2968–2977, 2012.

N.G. Hall and M.E. Posner. Earliness-tardiness scheduling problems, I: Weighted deviation of completion times about a common due date. *Operations Research*, 39:836–846, 1991.

M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10:196–210, 1962.

E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21:277–292, 1974.

K. Jansen, F. Land, and K. Land. Bounding the running time of algorithms for scheduling and packing problems. In F. Dehne, R. Solis-Oba, and J.R. Sack, editors, *Algorithms and Data Structures. WADS 2013.*, pages 439–450. Springer, vol 8037, 2013a.

K. Jansen, F. Land, and K. Land. Bounding the running time of algorithms for scheduling and packing problems. In F. Dehne, R. Solis-Oba, and J.-R. Sack, editors, *Algorithms and Data Structures*, volume 8037 of *Lecture Notes in Computer Science*, pages 439–450. Springer Berlin Heidelberg, 2013b.

J. Jozefowska. *Just-in-Time Scheduling: Models and algorithms for computer and manufacturing systems*. Springer, 2007.

J.J. Kanet. Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly*, 28(4):643–651, 1981.

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

C. Lenté, M. Liedloff, A. Soukhal, and V. T'kindt. Exponential algorithms for scheduling problems. hal.archives-ouvertes.fr/hal-00944382v1, 2014.

L. Shang, C. Lenté, M. Liedloff, A. Soukhal, and V. T'kindt. Exact exponential algorithms for 3-machine flowshop scheduling problems. *Journal of Scheduling*, doi.org/10.1007/s10951-017-0524-2, 2017.

L. Shang, M. Garraffa, F. Della Croce, and V. T'kindt. An exact exponential branch-and-merge algorithm for the single machine total tardiness problem. *Theoretical Computer Science*, 2018.

V. T'kindt and J.-C. Billaut. *Multicriteria scheduling: Theory, models and algorithms*. Springer, 2nd edition, 2006.

G. Woeginger. Space and time complexity of exact algorithms: Some open problems. In R. Downey, M. Fellows, and F. Dehne, editors, *Parameterized and Exact Computation - 1st International Workshop, IWPEC 2004, Proceedings*, pages 281–290. Springer, vol 3162, 2004.

G.J. Woeginger. Exact algorithms for NP-hard problems: A survey. pages 185–207. Lecture Notes in Computer Science, vol 2570, Springer, 2003.