Happach, Felix

Article — Published Version

Makespan minimization with OR-precedence constraints

Journal of Scheduling

Provided in Cooperation with:
Springer Nature

Suggested Citation: Happach, Felix (2021) : Makespan minimization with OR-precedence constraints, Journal of Scheduling, ISSN 1099-1425, Springer US, New York, NY, Vol. 24, Iss. 3, pp. 319-328, https://doi.org/10.1007/s10951-021-00687-6

Journal of Scheduling (2021) 24:319–328
https://doi.org/10.1007/s10951-021-00687-6

Makespan minimization with OR-precedence constraints

Felix Happach[1]

Accepted: 27 April 2021 / Published online: 29 May 2021

Abstract
We consider a variant of the NP-hard problem of assigning jobs to machines to minimize the completion time of the last job. Usually, precedence constraints are given by a partial order on the set of jobs, and each job requires all its predecessors to be completed before it can start. In this paper, we consider a different type of precedence relation that has not been discussed as extensively and is called OR-precedence. In order for a job to start, we require that *at least one* of its predecessors is completed—in contrast to *all* its predecessors. Additionally, we assume that each job has a release date before which it must not start. We prove that a simple List Scheduling algorithm due to Graham (Bell Syst Tech J 45(9):1563–1581, 1966) has an approximation guarantee of 2 and show that obtaining an approximation factor of $4/3 - \varepsilon$ is NP-hard. Further, we present a polynomial-time algorithm that solves the problem to optimality if preemptions are allowed. The latter result is in contrast to classical precedence constraints where the preemptive variant is already NP-hard. Our algorithm generalizes previous results for unit processing time jobs subject to OR-precedence constraints, but without release dates. The running time of our algorithm is $O(n^2)$ for arbitrary processing times and it can be reduced to $O(n)$ for unit processing times, where n is the number of jobs. The performance guarantees presented here match the best-known ones for special cases where classical precedence constraints and OR-precedence constraints coincide.

Keywords Scheduling · Precedence constraints · Approximation algorithm · Makespan

1 Introduction

In this paper, we consider the problem of scheduling jobs with OR-precedence constraints on parallel identical machines to minimize the time necessary to complete all jobs. Let $[n] := \{1, \dots, n\}$ be the set of jobs and m be the number of machines. Each job $j \in [n]$ is associated with a nonnegative integer processing time $p_j \in \mathbb{N}_0$ and a nonnegative integer release date $r_j \in \mathbb{N}_0$. The precedence constraints are given by a directed graph $G = ([n], E)$. The set of *predecessors* of a job $j \in [n]$ is $\mathscr{P}(j) = \{i \in [n] \mid (i, j) \in E\}$.

A *schedule* is an assignment of the jobs in $[n]$ to the machines such that each job j is processed by a machine for p_j units of time and each machine processes only one job at a time. Depending on the problem definition, jobs are allowed to be preempted at integer points in time (*preemptive scheduling*) or not at all (*non-preemptive scheduling*). The *start time* and *completion time* of job $j \in [n]$ are denoted by S_j and C_j, respectively. Note that $C_j \geq S_j + p_j$ and equality holds if job $j \in [n]$ is not preempted.

A schedule is called *feasible* if $S_j \geq \min\{C_i \mid i \in \mathscr{P}(j)\}$ and $S_j \geq r_j$ for all jobs $j \in [n]$. A job without predecessors may start at any point in time $t \geq r_j$. In other words, every job with predecessors requires that *at least one* of its predecessors is completed before it can start, and no job may start before it gets released. A job j is called *available* at time $t \geq 0$ if $t \geq r_j$ and, unless $\mathscr{P}(j) = \emptyset$, there is $i \in \mathscr{P}(j)$ with $C_i \leq t$. Our goal is to determine a feasible schedule that minimizes the *makespan*, which is defined as $C_{\max} := \max_{j \in [n]} C_j$. In an extension of the notation in Johannes (2005) and the three-field notation of Graham et al. (1979), the preemptive and non-preemptive variant of this problem are denoted by $P \mid r_j, \, or\text{-}prec, \, pmtn \mid C_{\max}$ and $P \mid r_j, \, or\text{-}prec \mid C_{\max}$, respectively.

Note that any job with zero processing time may be disregarded, so we assume, w.l.o.g., from now on that $p_j > 0$ for all jobs $j \in [n]$. As discussed below, the non-preemptive problem is NP-hard, which is why we are interested in

✉ Felix Happach
felix.happach@tum.de

[1] Department of Mathematics, School of Management, Technische Universität München, Munich, Germany

approximation algorithms. Let Π be a minimization problem, and $\alpha \geq 1$. Recall that an α-approximation algorithm for Π is a polynomial-time algorithm that returns a feasible solution with objective value at most α times the optimal objective value.

1.1 Non-preemptive scheduling

Garey and Johnson (1978) proved that the non-preemptive variant is already strongly NP-hard in the absence of precedence constraints and release dates. It remains NP-hard, even if the number of machines is fixed to $m = 2$ (Lenstra et al. 1977). In his seminal paper, Graham (1966) showed that a simple algorithm called *List Scheduling* achieves an approximation guarantee of 2:

> Consider the jobs in arbitrary order. Whenever a machine is idle, execute the next available job in the order on this machine. If there is no available job, then wait until a job completes.

If the jobs are sorted in order of non-increasing processing times, then List Scheduling is a $\frac{4}{3}$-approximation (Graham 1969). Hochbaum and Shmoys (1988) presented a $(1+\varepsilon)$-approximation for $P\,||\,C_{\max}$, which was improved in running time to the currently best-known by Jansen (2010). Mnich and Wiese (2015) showed that $P\,||\,C_{\max}$ is fixed parameter tractable with parameter $\max_{j\in[n]} p_j$. For non-trivial release dates, List Scheduling with an arbitrary job order is a 2-approximation algorithm (Hall and Shmoys 1989), and it is $\frac{3}{2}$-approximate if the jobs are sorted in order of non-increasing processing times (Chen and Vestjens 1997). Hall and Shmoys (1989) provided a $(1+\varepsilon)$-approximation for $P\,|\,r_j\,|\,C_{\max}$.

In contrast to the OR-precedence constraints that are considered in this paper, the standard precedence constraints, where each job requires that *all* its predecessors are completed, will be called *AND-precedence constraints*. Minimizing the makespan with AND-precedence constraints is strongly NP-hard, even if the number of machines is fixed to $m = 2$ and the precedence graph consists of disjoint paths (Du et al. 1991). List Scheduling is still 2-approximate in the presence of AND-precedence constraints if the order of the jobs is consistent with the precedence constraints (Graham 1966, 1969). The approximation factor can also be preserved for non-trivial release dates (Hall and Shmoys 1989). Assuming a variant of the Unique Games Conjecture (Khot 2002) together with a result of Bansal and Khot (2009), Svensson (2010) proved that this is essentially best possible.

If the precedence constraints are of AND/OR-structure[1] and the precedence graph is acyclic, then the problem without release dates still admits a 2-approximation algorithm (Gillies and Liu 1995). Erlebach et al. (2003) showed that the assumption on the precedence graph is not necessary. Both results first transform the instance to an AND-precedence constrained instance by fixing a predecessor of the OR-precedence constraints. Then, they solve the resulting instance with AND-precedence constraints using List Scheduling.

1.2 Preemptive scheduling

If preemptions are allowed, the algorithm of McNaughton (1959) computes an optimal schedule in the absence of release dates and precedence constraints. Ullman (1975) showed that the problem with AND-precedence constraints is NP-hard, even if all jobs have unit processing time. Lenstra and Rinnooy Kan (1978) proved that the problem of minimizing the makespan with AND-precedence constraints and unit processing time jobs cannot be approximated better than 4/3, unless P = NP. Note that if $p_j = 1$ for all jobs j, then there is no benefit in preemption. This implies that the hardness results of the unit processing time jobs carry over to the preemptive case. However, the preemptive variant becomes solvable in polynomial time for certain restricted precedence graphs. Precedence graphs that consist of outtrees are of special interest to us, since then AND- and OR-precedence constraints coincide.

A number of polynomial-time algorithms were proposed for AND-precedence constraints in form of an outtree. Hu (1961) proposed the first such algorithm for unit processing time jobs, and Brucker et al. (1977) presented an algorithm that can also deal with non-trivial release dates, which was improved in running time by Monma (1982). Muntz and Coffman (1970) gave a polynomial-time algorithm for the preemptive variant. The algorithm of Gonzalez and Johnson (1980) has an asymptotically better running time and uses fewer preemptions than the one in Muntz and Coffman (1970). Finally, Lawler (1982) proposed a polynomial-time algorithm for the preemptive variant that can deal with non-trivial release dates, if the precedence graph consists of outtrees. For general OR-precedence constraints and unit processing time jobs, Johannes (2005) presented a polynomial-time algorithm that is similar to Hu's algorithm (Hu 1961).

[1] That is, the set of jobs can be partitioned into those jobs that require *all* of their predecessors to be completed before they can start ("AND-jobs") and those jobs that require *at least one* of their predecessors to be completed ("OR-jobs").

1.3 Main results

Our first result shows that the makespan of *every* feasible schedule without unnecessary idle time on the machines is at most twice the optimal makespan, even if non-trivial release dates are involved.

Theorem 1 *List Scheduling is a* $\left(2-\frac{1}{m}\right)$*-approximation for* $P\,|\,r_j,\ or\text{-}prec\,|\,C_{\max}$.

The proof of Theorem 1 is contained in Sect. 3. Using a reduction from the well-known VERTEX COVER problem, we provide a lower bound on the approximability of scheduling with OR-precedence constraints. This lower bound coincides with the corresponding lower bound for AND-precedence constraints presented in Lenstra and Rinnooy Kan (1978).

Theorem 2 *It is NP-hard to approximate* $P\,|\,or\text{-}prec\,|\,C_{\max}$ *within a factor* $\frac{4}{3}-\varepsilon$ *for any* $\varepsilon>0$.

For $P\,|\,r_j,\ or\text{-}prec,\ pmtn\,|\,C_{\max}$, we improve on the result of Johannes (2005) by analyzing the structure of an optimal solution. The key ingredient is the concept of *minimal chains* that we introduce in Sect. 2. Informally the length of the minimal chain of job $j\in[n]$ is the minimal amount of time that we need to complete j. The minimal chain of j is the set of jobs that have to be processed in order to complete j in that time. We show that there is an optimal preemptive schedule where each job is preceded by its minimal chain. We then exploit this structure to transform the instance into an equivalent AND-precedence constrained instance, where we can apply the algorithm of Lawler (1982). Thereby, we obtain our third result. The proof is contained in Sect. 4.

Theorem 3 $P\,|\,r_j,\ or\text{-}prec,\ pmtn\,|\,C_{\max}$ *can be solved to optimality in time* $O(n^2)$.

Since there is no need to preempt if $p_j=1$ for all $j\in[n]$ and we can use the algorithm of Monma (1982) instead of Lawler (1982), we immediately obtain the following corollary. This generalizes the aforementioned result of Johannes (2005) by also incorporating release dates.

Corollary 1 $P\,|\,r_j,\ or\text{-}prec,\ p_j=1\,|\,C_{\max}$ *can be solved to optimality in time* $O(n)$.

We would like to remark that Corollary 1 without release dates was already proved by Johannes (2005). However, the size of the preemptive instance is not polynomial in the input parameters of the initial instance. Thus, the analysis in Johannes (2005) cannot be extended to the preemptive case.

2 Preliminaries and minimal chains

In order to simplify some arguments, we introduce a dummy job s with $p_s=r_s=0$ that precedes all jobs. That is, we assume that the set of jobs is $N=[n]\cup\{s\}$ and introduce an arc (s,j) for all $j\in[n]$ with $\mathscr{P}(j)=\emptyset$ in the precedence graph G. A set $S\subseteq N$ is called a *feasible starting set* if all jobs in S are reachable from s in the induced precedence subgraph $G[S\cup\{s\}]$. The set of feasible starting sets is denoted by $\mathscr{S}$. Note that there is a feasible schedule if and only if $N\in\mathscr{S}$, i.e., the complete set of jobs is a feasible starting set. In particular, we can decide in linear time, e.g., via breadth-first-search, whether there exists a feasible schedule. Henceforth, we will assume that the instances we consider admit a feasible schedule.

Note that $P\,|\,or\text{-}prec\,|\,C_{\max}$ is a generalization of $P\,||\,C_{\max}$, which is already strongly NP-hard (Garey and Johnson 1978). If G is an outtree rooted at s, then OR- and AND-precedence constraints are equivalent. The NP-hardness result of Du et al. (1991) implies that the problem remains strongly NP-hard, even if the number of machines is fixed.

Observation 4 $Pm\,|\,or\text{-}prec\,|\,C_{\max}$ *is strongly NP-hard for all* $m\geq 2$.

In order to analyze the performance of our algorithms, we use the concept of what we call *minimal chains*. Informally, a minimal chain of a job k is a set of jobs that need to be scheduled so that k can complete as early as possible. To define minimal chains properly, we use the notion of an *earliest start schedule*, see, e.g., (Erlebach et al. 2003; Möhring et al. 2004; Johannes 2005). Although these schedules are well-defined for general AND/OR-precedence constraints, we only need and define them in the OR-context.

The *earliest start schedule* is defined as a schedule on an infinite number of machines such that a job j without predecessors starts at time r_j and a job j with $\mathscr{P}(j)\neq\emptyset$ starts at time $\max\{r_j,\min\{C_i\,|\,i\in\mathscr{P}(j)\}\}$. Clearly, an earliest start schedule respects the OR-precedence constraints of the instance, since every job is preceded by at least one of its predecessors. Also, the completion time of a job in any feasible schedule on m machines is bounded from below by its completion time in the earliest start schedule. That is, if C_j denotes the completion time of job j in the earliest start schedule, the optimum makespan satisfies $C^*_{\max}\geq\max\{C_j\,|\,j\in N\}$. Note that an earliest start schedule is not necessarily unique, but the start and completion times of all jobs are fixed. Earliest start schedules can be constructed in polynomial time by iteratively scheduling every job as early as possible (Erlebach et al. 2003).

Let $k\in N$ and let C_j be the completion time of $j\in N$ in the earliest start schedule. A set $L\subseteq N$ is called a *minimal chain of* k if $L\in\mathscr{S}$ is an inclusion-minimal feasible starting set such that $k\in L$ and $\max_{j\in L}C_j=C_k$. The set of minimal chains of k is denoted by $\mathscr{MC}(k)$, and the *length of the minimal chain of* k is $mc(k):=C_k$.

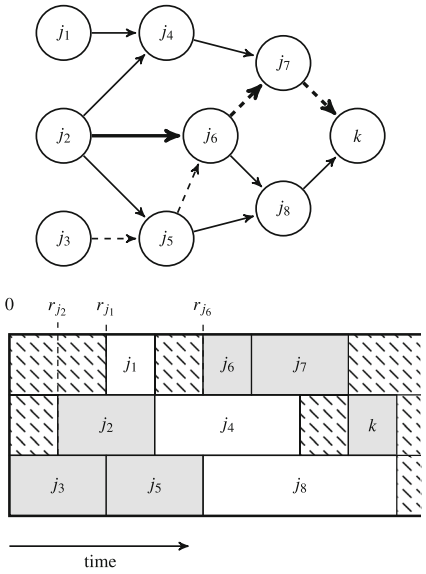

Fig. 1 An instance on nine jobs with processing times $p_{j_1} = p_{j_6} = p_k = 1$, $p_{j_2} = p_{j_3} = p_{j_5} = p_{j_7} = 2$, $p_{j_4} = 3$, $p_{j_8} = 4$ and release dates $r_{j_1} = 2$, $r_{j_2} = 1$, $r_{j_6} = 4$, $r_j = 0$ for all other jobs j (top) and an earliest start schedule (bottom). The set of minimal chains of k is $\mathscr{MC}(k) = \{\{j_2, j_6, j_7, k\}, \{j_3, j_5, j_6, j_7, k\}\}$ with $mc(k) = 8$. The chain $\{j_2, j_6, j_7, k\}$ is dominated by j_6, and $\{j_3, j_5, j_6, j_7, k\}$ is dominated by jobs j_3 and j_6. The paths in G that correspond to the minimal chains in $\mathscr{MC}(k)$ are depicted dashed and thick, respectively. Dashed thick arcs correspond to both minimal chains. Jobs in minimal chains are highlighted in gray

We can construct a minimal chain of k by iteratively tracing back predecessors that delay job k in the earliest start schedule. That is, starting at k, we mark one of its predecessors j with $C_j = S_k$, and then proceed with j in the same manner, i.e., we mark a predecessor i of j with $C_i = S_j$, and so on, until we reach a job i' that starts at its release date. If i' has no predecessors, we are done. If $\mathscr{P}(i') \neq \emptyset$, we mark a predecessor j' of i' with $C_{j'} \leq S_{i'}$ and continue with j' as described above. The marked jobs now correspond to a minimal chain of k. That is, a minimal chain $L = \{j_1, \ldots, j_\ell\} \in \mathscr{MC}(k)$ is a path in G with $\mathscr{P}(j_1) = \emptyset$, $j_q \in \mathscr{P}(j_{q+1})$ for all $q \in [\ell - 1]$ and $j_\ell = k$ such that $S_{j_1} = r_{j_1}$ and $S_{j_q} = \max\{r_{j_q}, C_{j_{q-1}}\}$ for all $2 \leq q \leq \ell$. We call j_q *the predecessor of* j_{q+1} in L for $q \in [\ell - 1]$ and denote this by $\mathscr{P}_L(j_{q+1}) := \{j_q\}$. A job $j_h \in L$ is said to *dominate* the minimal chain L if $mc(k) = r_{j_h} + \sum_{q=h}^{\ell} p_{j_q}$. Figure 1 illustrates an example.

In the following, we denote the completion times in an optimal schedule by C_j^* (for $j \in [n]$) and its makespan by $C_{\max}^*$. Also, we sometimes denote an optimal schedule by C^* and the schedule with completion times C_j (for $j \in [n]$) by C. There are two trivial lower bounds on the optimal makespan. First, any feasible schedule cannot do better than splitting the total processing load equally among all machines, so $C_{\max}^* \geq \frac{1}{m} \sum_{j \in N} p_j$. Second, every job requires at least one of its predecessors to be completed before it can start. If we start with an empty schedule, the earliest completion time of job j is by definition equal to the length of its minimal chain. Thus, $C_{\max}^* \geq \max_{j \in N} mc(j)$.

3 Approximability and hardness for the non-preemptive setting

Erlebach et al. (2003) presented a 2-approximation algorithm for minimizing the makespan with AND/OR-precedence constraints. The algorithm transforms the instance to an AND-instance by fixing an OR-predecessor for each job, and then applies List Scheduling. We show that also without transforming the instance, List Scheduling is 2-approximate for OR-precedence constraints, even with non-trivial release dates. The proof idea is similar to Hall and Shmoys (1989). Since we consider OR-precedence constraints, we need the notion of minimal chains to bound the amount of idle time on the machines. The following lemma proves Theorem 1.

Lemma 1 *List Scheduling is a* $\left(2 - \frac{1}{m}\right)$*-approximation for* $P \,|\, r_j,\ or\text{-}prec \,|\, C_{\max}$.

Proof Consider the schedule returned by List Scheduling, and let S_j and C_j be the start and completion time of job $j \in [n]$. Let $k \in [n]$ be a job that completes last, i.e., $C_k = C_{\max}$. Let $I \subseteq [0, S_k]$ be the union of all time intervals where some machine is idle. If $I = \emptyset$, then all machines are busy before time S_k with jobs in $N \setminus \{k\}$. Hence,

$$C_{\max} = S_k + p_k \leq \frac{1}{m} \sum_{j \neq k} p_j + p_k$$
$$= \frac{1}{m} \sum_{j \in N} p_j + \left(1 - \frac{1}{m}\right) p_k \leq \left(2 - \frac{1}{m}\right) C_{\max}^*.$$

So suppose there is idle time, and let I be the union of all intervals in which some machine is idle. Let $S \subseteq N$ be a set of jobs such that S is a path in the precedence graph from the source s to k. At every point in time $t \in I$, a job in S is either not yet released, or is currently running on some machine. Otherwise, there is an unscheduled available job in S that can be processed at time t. Let $L' \in \mathscr{MC}(k)$ be a minimal chain of k and enumerate the jobs $L' = \{j_1, \ldots, j_\ell\}$ such that $j_\ell = k$, $\mathscr{P}(j_1) = \emptyset$, and $j_q \in \mathscr{P}(j_{q+1})$ for all $q \in [\ell - 1]$. Recall that L' is a path in G, so, at every idle point in time, some job of L' is either being processed or not yet released. That is, the total idle time is $|I| \leq mc(k)$, but we can even get an even stronger bound.

Let $h \in [\ell]$ be maximal such that j_h dominates the minimal chain L', i.e., $mc(k) = r_{j_h} + \sum_{q=h}^{\ell} p_{j_q}$. Let $L := \{j_h, \ldots, j_\ell\}$, and consider the points in time $I_L := [0; r_{j_h}] \cup \bigcup_{j \in L} [S_j; C_j]$ when j_h is not yet released or some job in L is being processed. Note that $|I_L| \leq r_{j_h} + \sum_{j \in L} p_j$.

W.l.o.g., we can assume that at least one machine is running during $[0; r_{j_h}]$. Otherwise, if all machines were idle at some point $t \in [0; r_{j_h}]$, then all jobs j with $r_j \leq t$ are already completed at time t. Thus, also in the optimum solution, no machine is running at time t, so we can disregard these time slots where no machine is running at all. That is, during $I_B := [0; C_{\max}] \setminus I_L$, all machines are busy with jobs in $N \setminus L$ and the total processing load of jobs that are running in I_B is less or equal than $\sum_{j \notin L} p_j - r_{j_h}$. Hence, $|I_B| \leq \frac{1}{m}(\sum_{j \notin L} p_j - r_{j_h})$ and we obtain

$$C_{\max} = |I_B| + |I_L| \leq \frac{1}{m}\left(\sum_{j \notin L} p_j - r_{j_h}\right) + r_{j_h} + \sum_{q=h}^{\ell} p_{j_q}$$
$$= \frac{1}{m}\sum_{j \in N} p_j + \left(1 - \frac{1}{m}\right) mc(k) \leq \left(2 - \frac{1}{m}\right) C^*_{\max}.$$

This proves the claim. □

Corollary 2 *List Scheduling solves* $1 \,|\, r_j,\ or\text{-}prec \,|\, C_{\max}$ *to optimality.*

In the remainder of this section, we provide an inapproximability result for $P \,|\, or\text{-}prec \,|\, C_{\max}$, assuming P $\neq$ NP, and prove Theorem 2. Recall the definition of the NP-complete Vertex Cover problem (Karp 1972): Let $\mathcal{G} = (V, \mathcal{E})$ be an undirected graph. A *vertex cover* is a subset $W \subseteq V$ such that every edge in $\mathcal{E}$ is incidence to a vertex in W. The Vertex Cover problem asks whether, for given $K \in \mathbb{N}$, there is a vertex cover of size at most K.

To prove Theorem 2, we first describe a reduction from Vertex Cover to $P \,|\, or\text{-}prec \,|\, C_{\max}$. Then, we show that the instance of Vertex Cover is a YES-instance if and only if the corresponding instance of OR-scheduling has an optimum makespan of 3 (Lemma 2). Hence, if we had an α-approximation algorithm for $P \,|\, or\text{-}prec \,|\, C_{\max}$ with $\alpha < \frac{4}{3}$, we could use this algorithm to obtain a feasible schedule with makespan strictly less than 4. Since all input data in the instance are integers, the makespan of any feasible schedule is also integer. So, a feasible solution with makespan strictly less than 4 actually has makespan ≤ 3, i.e., the solution is optimal. Thus, we could use the α-approximation algorithm to find an optimum solution and decide whether the initial Vertex Cover instance is a YES-instance, implying P = NP.

We assume $1 \leq K \leq |V| - 1$, as otherwise Vertex Cover is trivial. If $\mathcal{G}$ is not connected, we can restrict the problem to finding a minimum vertex cover in each of the connected components. So, w.l.o.g., we can assume that $|V| \leq |\mathcal{E}|$, which is true if $\mathcal{G}$ is connected and contains a cycle. In fact, the only connected undirected graphs with $|V| > |\mathcal{E}|$ are *trees*, for which Vertex Cover can be solved in polynomial time using a simple greedy algorithm.

We now describe the construction of the instance of $P \,|\, or\text{-}prec \,|\, C_{\max}$. The number of machines is $m = |\mathcal{E}| + |V| - K$, and the set of jobs consists of four different sets, $J_K \cup J_V \cup J_{\mathcal{E}} \cup X$. We refer to Fig. 2 for an instance of Vertex Cover and the corresponding OR-scheduling instance.

In J_K, we introduce K jobs of unit processing time. The sets J_V and $J_{\mathcal{E}}$ contain a job j_v and $j_{\{v,u\}}$ for every vertex $v \in V$ and every edge $\{v, u\} \in \mathcal{E}$, respectively. The processing times of all jobs $j_v \in J_V$ and $j_{\{v,u\}} \in J_{\mathcal{E}}$ are equal to 1. We set the predecessor of an edge-job in $J_{\mathcal{E}}$ to be the jobs corresponding to its incident vertices, i.e., $\mathcal{P}(j_{\{v,u\}}) = \{j_v, j_u\}$ for all $\{v, u\} \in \mathcal{E}$. Moreover, we assign OR-precedence constraints in $J_K \times J_V$ in the form of an outforest such that each job in J_V is successor of exactly one job in J_K. This can be done since $|J_K| = K \leq |V| = |J_V|$. It is not important which job in J_K is the predecessor of which job in J_V. We only need these precedence constraints to ensure that no job in J_V can start at time 0 and all jobs in J_V are available as soon as all jobs in J_K are completed. The remaining jobs in X are dummy jobs to enforce a certain structure of any optimal schedule. The set X contains $m - K$ jobs of processing time equal to 2, and these jobs do not have any predecessors or successors. The next lemma together with the previous discussion completes the proof of Theorem 2.

Lemma 2 *Let* $\mathcal{G} = (V, \mathcal{E})$ *be an undirected graph and* $K \in [|V| - 1]$. Vertex Cover *is a YES-instance iff the corresponding instance of* $P \,|\, or\text{-}prec \,|\, C_{\max}$ *has makespan* ≤ 3.

Proof Suppose Vertex Cover is a YES-instance. Let $W \subseteq V$ be a vertex cover of size $|W| \leq K$, and let $J_W \subseteq J_V$ be the jobs corresponding to W. We can schedule the jobs in a similar structure as in Fig. 2 (right). That is, schedule all jobs in X and J_K in the intervals $[0; 2]$ and $[0; 1]$, respectively. So all jobs in J_V are available at time 1. In $[1; 2]$, there are exactly $m - |X| = K$ slots left in which we can schedule the jobs in J_W and some other jobs of $J_V \setminus J_W$ if $|W| < K$. Hence, at time 2, all jobs in $J_{\mathcal{E}}$ are available and can be scheduled in $[2; 3]$. Finally, we schedule the remaining jobs in J_V on the remaining $m - |\mathcal{E}| = |V| - K$ machines in the interval $[2; 3]$.

Now suppose that the instance of $P \,|\, or\text{-}prec \,|\, C_{\max}$ has makespan ≤ 3. Recall that $m = |\mathcal{E}| + |V| - K$ and note that the total processing load of all jobs is

$$\sum_{j \in N} p_j = |J_K| + |J_V| + |J_{\mathcal{E}}| + 2\,|X|$$
$$= K + |V| + |\mathcal{E}| + 2\,(m - K) = 3m. \qquad (1)$$

So any feasible schedule with makespan ≤ 3 has makespan equal to 3, and there is no idle time within the interval $[0; 3]$. Consider an optimal schedule of makespan equal to 3. Note that, due to the precedence constraints, no

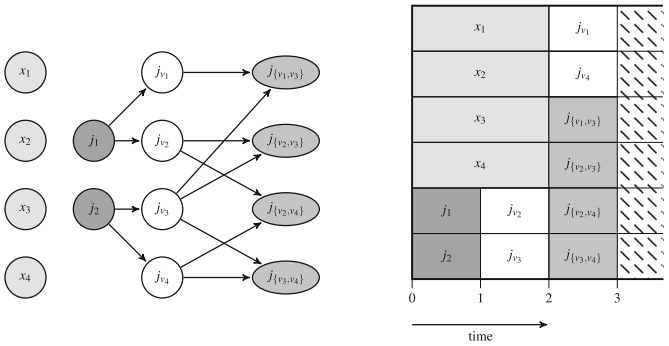

Fig. 2 The precedence graph of the instance of $P \mid or\text{-}prec \mid C_{\max}$ in the reduction from VERTEX COVER for $\mathscr{G} = (\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_3\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}\})$ and $K = 2$ (left) and a feasible schedule with makespan ≤ 3 (right). To highlight the structure of the schedule, jobs in J_K, $J_{\mathscr{E}}$ and X are depicted in different shades of gray

job in J_V can start before time 1, and no job in $J_{\mathscr{E}}$ can start before time 2. Thus, all jobs in $J_{\mathscr{E}}$ have to be scheduled in the interval $[2; 3]$. Also, the jobs of J_V that are scheduled in $[1; 2]$ correspond to a vertex cover, since otherwise not all jobs in $J_{\mathscr{E}}$ would be available at time 2. It remains to be shown that the set of jobs of J_V scheduled in $[1; 2]$ has cardinality at most K.

The total processing load of all jobs is equal to $3m$, see (1). So there cannot be idle time within $[0; 3]$, i.e., m jobs start at time 0. The only jobs that can start at time 0 are those in $J_K \cup X$ (all other jobs have predecessors). Since $|J_K| + |X| = K + m - K = m$, we know that the jobs in J_K and X are scheduled in $[0; 1]$ and $[0; 2]$, respectively. This leaves exactly $m - |X| = K$ slots in $[1; 2]$ in which only jobs of J_V are scheduled. □

Technically, it would suffice to let J_K be a singleton that precedes all jobs in J_V. However, choosing $|J_K| = K$, and, thus, the total processing load to be equal to $3m$ makes the argument slightly easier. Note that the processing times in the OR-scheduling instance in the reduction are in $\{1, 2\}$ only. This is indeed necessary, i.e., we cannot get a reduction for unit processing times. Also, the proof does not work if we allow preemption. In this case, we could preempt a job in X at time 1, and process more than K jobs in the interval $[1; 2]$. In fact, as we see in the next section, we can solve the variants with unit processing times or preemption in polynomial time.

4 A polynomial-time algorithm for the preemptive case

In this section, we consider $P \mid r_j, or\text{-}prec, pmtn \mid C_{\max}$ and prove Theorem 3. Recall that all processing times and release dates of jobs in $[n]$ are positive and nonnegative integers, respectively. So preemptive scheduling and non-preemptive scheduling of unit processing time jobs are equivalent, since there is no need to preempt, which proves Corollary 1.

In contrast to the non-preemptive instance, an optimal preemptive schedule will never have idle time if there are available jobs. Without preemption, it could make sense to wait for some job j to finish, i.e., have idle time, although there is an available job k. The reason might be that we want to process a successor i of j right away. However, if we allow preemption, then we could just schedule a fraction of k, and once j completes, we preempt k and process i.

We first derive some necessary notions, and then present a polynomial-time algorithm that computes an optimal preemptive schedule. Fix $L_j \in \mathscr{MC}(j)$ for all $j \in N$. The collection of minimal chains $\{L_j \mid j \in N\}$ is called *closed* if $i \in L_j$ implies $L_i \subseteq L_j$ for all $j \in N$. Note that we can always choose $L_i \subseteq L_j$ for all $i \in L_j$, since (informally) subpaths of shortest paths are shortest paths. Hence, if we compute minimal chains $L_1, \ldots, L_n$ using the procedure described in Sect. 2, we may assume that $\{L_1, \ldots, L_n\}$ is closed. We say an arc $(i, j) \in E$ is *in line with the minimal chain* L_j if $i \in L_j$. Recall that all processing times are strictly positive and $L_j \in \mathscr{MC}(j)$. So if $(i, j) \in E$ is in line with L_j, then $i \in \mathscr{P}(j)$.

Our algorithm, which we refer to as ALGOPMTN, works as follows and is summarized in Algorithm 1. First, compute a closed collection of minimal chains $\{L_j \mid j \in N\}$. Then, transform the instance to an instance with AND-precedence constraints by deleting all arcs that are not in line with $L_1, \ldots, L_n$ and denote the resulting graph by G'. Note that G' is an outtree. Now, apply a polynomial-time algorithm for the resulting AND-instance to compute an optimal preemptive schedule. (Recall that we can compute optimal preemptive schedules for these special cases in polynomial time, see, e.g., Hu 1961; Muntz and Coffman 1970; Brucker et al. 1977; Gonzalez and Johnson 1980; Monma 1982; Lawler 1982. We use the algorithm of Lawler (1982), but instead, depending on the setting, we could also use any of the other algorithms.)

We prove that ALGOPMTN works correctly by analyzing the structure of an optimal preemptive schedule. More precisely, we show that for any closed collection of minimal chains, there is an optimal preemptive schedule that is feasible for the transformed graph G'. Before we are able to prove Theorem 3, we need some additional notation.

Input: Instance of $P \mid r_j,\ pmtn,\ or\text{-}prec \mid C_{\max}$
Output: A feasible schedule for $P \mid r_j,\ pmtn,\ or\text{-}prec \mid C_{\max}$

1 Construct an earliest start schedule;
2 Compute a closed collection $\{L_j \mid j \in N\}$ with $L_j \in \mathscr{MC}(j)$ for all $j \in N$;
3 $E' \leftarrow \emptyset$;
4 **for** $j \in N$ **do**
5 Enumerate the jobs in $L_j = \{j_1, \ldots, j_\ell\}$ so that $j_q \in \mathscr{P}_{L_j}(j_{q+1})$ for all $q \in [\ell - 1]$;
6 $E' \leftarrow E' \cup \{(j_1, j_2), (j_2, j_3), \ldots, (j_{\ell-1}, j_\ell)\}$;
7 **end**
8 Set $G' = (N, E')$;
9 Apply Lawler's algorithm (Lawler 1982) for $P \mid r_j,\ pmtn,\ prec = outtree \mid C_{\max}$ on G';
10 **return** schedule returned by Lawler's algorithm (Lawler 1982);

Algorithm 1: AlgoPmtn for $P \mid r_j,\ pmtn,\ or\text{-}prec \mid C_{\max}$.

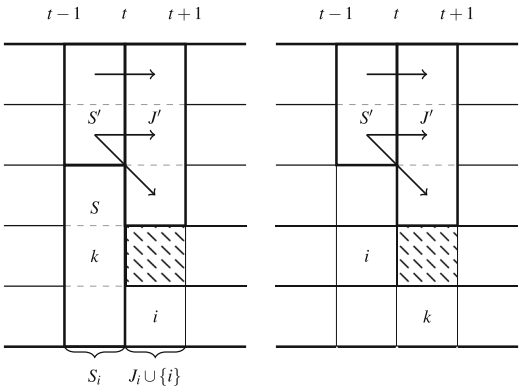

Fig. 3 Relevant time slots $[t-1; t+1]$ in the initial schedule (left) and final schedule (right), respectively. The arcs indicate that the respective job in S' is the predecessor of the corresponding job in J'. In this example, $J = \emptyset$, so $J_i = J'$

If jobs are allowed to preempt, we need to "keep track" of how much of the minimal chain of a job is already processed at every point in time. To formalize this, we split every job $j \in [n]$ into p_j jobs $j_1, \ldots, j_{p_j}$ of unit processing time. The predecessors of these jobs are $\mathscr{P}(j_1) = \{i_{p_i} \mid (i, j) \in E\}$ and $\mathscr{P}(j_u) = \{j_{u-1}\}$ for all $2 \le u \le p_j$. The release dates are $r_{j_u} = r_j$ for all $j \in N$ and $u \in [p_j]$. As before, we add a dummy job s with $p_s = r_s = 0$ and $\mathscr{P}(j_1) = \{s\}$ if $\mathscr{P}(j) = \emptyset$ for $j \in [n]$. We refer to this instance as the *preemptive instance* and denote the set of jobs by $N^{(p)}$.

Note that $N^{(p)} = N$ if all jobs have unit processing time. We informally extend definition of $mc(k)$ to *fractions of jobs* via the original definition on the preemptive instance. Note that (the lengths of) all minimal chains coincide with the non-preemptive instance. In particular, all lower bounds on the makespan are still valid, and $i \in L_j$ implies $i_1, \ldots, i_{p_i} \in L_{j_u}$ for all $u \in [p_j]$. Since minimal chains in the non-preemptive and preemptive instance coincide, $\{L_j \mid j \in N^{(p)}\}$ is closed iff $\{L_j \mid j \in N\}$ is closed. Two distinct jobs $i, j \in N^{(p)}$ are called *inverted* w.r.t. the closed collection of minimal chains $\{L_k \mid k \in N^{(p)}\}$ in the schedule C if $i \in L_j$ and $C_i \ge C_j$. Let I_C be the number of inversions in the schedule C.

Lemma 3 describes a procedure that swaps two jobs $i, k \in N^{(p)}$ that are scheduled consecutively. We apply this procedure to show that there always exists an optimal solution without inversions (see Lemma 4). For the notation of Lemma 3, we forget about release dates, i.e., consider schedules for $P \mid or\text{-}prec,\ pmtn \mid C_{\max}$. We describe how to incorporate release dates in the proof of Lemma 4, which is the key lemma for the correctness of AlgoPmtn.

Lemma 3 *Let $\{L_j \mid j \in N^{(p)}\}$ be a closed collection of minimal chains and C^* be a feasible preemptive schedule. Let $i \in N^{(p)}$ with $C^*_i \ge 2$, and let $S_i = \{j \in N^{(p)} \mid C^*_j = C^*_i - 1\}$ be the jobs scheduled directly before i. Assume that $|S_i| = m$ and $C^*_j \le C^*_i - 2$ for $j \in \mathscr{P}_{L_i}(i)$.*[2] *Then, there is $k \in S_i$ such that swapping i and k, i.e., setting $C'_i = C^*_i - 1 = C^*_k$, $C'_k = C^*_k + 1 = C^*_i$ and $C'_j = C^*_j$ for all $j \in N^{(p)} \setminus \{i, k\}$, yields a feasible schedule with $C'_{\max} = C^*_{\max}$ and $I_{C'} \le I_{C^*}$.*

Proof To shorten notation, set $t := C^*_i - 1$. Note that the makespan does not change if we swap two unit processing time jobs. Let $J_i = \{j \in N^{(p)} \setminus \{i\} \mid C^*_j = t + 1\}$ be the jobs running in parallel to i on the other machines. Note that $|J_i| \le m - 1$, and recall that there are $|S_i| = m$ jobs that are being processed directly before i. For $j \in N^{(p)}$, let $\mathscr{A}^*_j := \{j' \in N^{(p)} \mid C^*_{j'} < C^*_j\}$ be the set of jobs that complete before j starts.

Let $J' = \{j \in J_i \mid \mathscr{P}_{L_j}(j) \cap S_i \ne \emptyset\} \cup \{j \in J_i \mid |\mathscr{P}(j) \cap \mathscr{A}^*_j| = 1 \text{ and } \mathscr{P}(j) \cap \mathscr{A}^*_j \subseteq S_i\}$ be the set of jobs that are scheduled parallel to i and that are processed directly after their predecessor in the minimal chain or the only predecessor preceding them in the schedule. Let $S' \subseteq S_i$ be the set of these predecessors of jobs in J'. We do not want to swap i with a job in S' since this would cause an inversion or yield an infeasible schedule. Note that $|S'| \le |J'|$, and set $S = S_i \setminus S'$ and $J = J_i \setminus J'$. Then, $|S| = m - |S'| \ge m - |J'| \ge |J_i| + 1 - |J'| = |J| + 1 \ge 1$, so $S \ne \emptyset$. Figure 3 illustrates the sets and the corresponding schedules before and after the swap. It is clear that any $k \in S$ satisfies the claim. □

Lemma 4 *Let $\{L_j \mid j \in N^{(p)}\}$ be a closed collection of minimal chains $L_j \in \mathscr{MC}(j)$ for all $j \in N^{(p)}$. There exists an optimal preemptive schedule C^* such that $C^*_i < C^*_j$ for all $j \in N^{(p)}$ and $i \in L_j \setminus \{j\}$.*

Proof Recall that all processing times of jobs in $N^{(p)}$ are equal to 1. Consider an optimal schedule with completion

[2] So moving i to $[C^*_i - 2; C^*_i - 1]$ does not violate its precedence constraints or cause an inversion.

times C_j^* for all $j \in N^{(p)}$ such that the number of inversions I_{C^*} is minimal among all optimal solutions. Suppose by contradiction that $I_{C^*} \geq 1$. We show how to construct a schedule with $C'_{\max} = C^*_{\max}$ and $I_{C'} < I_{C^*}$ using Lemma 3.

Since the schedule is optimal, we can assume that the dummy job s starts at time 0. Let $j \in N^{(p)}$ and $i \in \mathscr{P}_{L_j}(j)$ such that (i, j) is an inverted pair, i.e., $C_i^* \geq C_j^* \geq mc(j) \geq mc(i) + 1$. We reindex the jobs in $L_j = \{j_0, j_1, \ldots, j_\ell, j_{\ell+1}\} \in \mathscr{MC}(j)$ such that $s = j_0$, $i = j_\ell$, $j = j_{\ell+1}$ and $j_{q-1} \in \mathscr{P}_{L_j}(j_q)$ for all $q \in [\ell + 1]$. Note that $mc(j_{q-1}) + 1 \leq mc(j_q) \leq C^*_{j_q}$ for all $q \in [\ell + 1]$ and $mc(j_0) = mc(s) = 0$.

Using Lemma 3, we move the jobs $j_1, \ldots, j_\ell$ successively (in this order) to the front such that they complete at times $mc(j_1), \ldots, mc(j_\ell)$, respectively. For all $k \in N^{(p)}$ with non-trivial release date, it holds

$$C_k^* \geq mc(k) \geq r_k + p_k = r_k + 1. \tag{2}$$

So we can swap those jobs $k \in \{j_1, \ldots, j_\ell\}$ that do not complete at time $mc(k)$ to the front without violating the respective release dates. Thereby, we obtain a schedule that satisfies

$$0 = C'_{j_0} < mc(j_1) = C'_{j_1} < mc(j_2) = C'_{j_2} < \cdots < \\ < \cdots < mc(j_\ell) = C'_{j_\ell} < C'_j. \tag{3}$$

Since we first move job j_1 to the front, then j_2, and so on, we ensure that, when we apply Lemma 3 for $i = j_q$ (in the notation of Lemma 3), then its predecessor j_{q-1} completes at time $mc(j_{q-1}) < mc(j_q)$. So the assumptions of Lemma 3 are satisfied. The procedure of Lemma 3 does not violate any release dates, since $k \in S$ (in the notation of Lemma 3) is scheduled later and it is feasible to schedule j_q earlier due to (2) for all $q \in [\ell]$.

Figure 4 illustrates the current completion times and the time slots in which we move the jobs in the minimal chain L_j. Note that it is not necessary to move the job $j = j_{\ell+1}$. However, by applying Lemma 3, it might happen that $k = j$ (in the notation of Lemma 3) is chosen, i.e., j is "passively moved". Similarly, a job j_h might be "passively moved" when we swap j_q with $q < h$ to the front. This is not a problem, since we deal with j_h in a later iteration.

Multiple application of Lemma 3 ensures that the resulting schedule is feasible and has no more inversions than the initial schedule. Further, Lemma 3 implies $C'_{\max} = C^*_{\max}$, and $I_{C'} < I_{C^*}$ because i and j are not inverted anymore, see (3). This contradicts to the choice of the initial schedule being an optimal solution with fewest inversions. So there exists an optimal solution without inversions, which proves the claim. □

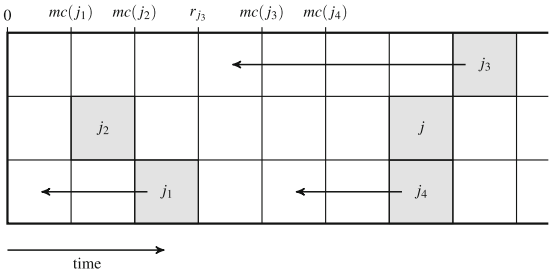

Fig. 4 Illustration of the procedure to move jobs in $L_j \setminus \{j\} = \{j_1, j_2, j_3, j_4\}$ to the front. Blank squares are jobs not in L_j. Arrows indicate into which time slot we want to move the respective jobs. The jobs are moved "lowest index first" rather than all at once. Note that $mc(j_3) > mc(j_2) + 1$ because $r_{j_3} = 3$

The following lemma shows correctness of ALGOPMTN and proves Theorem 3.

Lemma 5 ALGOPMTN *solves* $P \mid r_j, or\text{-}prec, pmtn \mid C_{\max}$ *to optimality in polynomial time. The running time of* ALGOPMTN *is* $O(n^2)$ *for arbitrary processing times and* $O(n)$ *for unit processing times.*

Proof First, observe that the graph G' constructed by ALGOPMTN is a subgraph of the initial precedence graph G. Since the schedule returned by the algorithm is feasible for the AND-instance on G' (this follows from correctness of Lawler's algorithm Lawler 1982), it certainly is feasible for the OR-instance on G. Construction of the earliest start schedule and Lawler's algorithm run in polynomial time (Erlebach et al. 2003; Lawler 1982). Also, we can compute the closed collection of minimal chains and construct G' in polynomial time. So, ALGOPMTN runs in polynomial time and returns a feasible schedule.

As for optimality of the schedule returned by ALGOPMTN, let $\{L_j \mid j \in N\}$ be the closed collection of minimal chains that is computed in the second step, and let G' be the corresponding subgraph of G. Since $\{L_j \mid j \in N\}$ is closed, G' is an outforest. Thus, OR- and AND-precedence constraints on G' are equivalent.

Consider the schedule returned by ALGOPMTN, i.e., by Lawler's algorithm (Lawler 1982) on G', and let $C_{\max}$ be its makespan. Since the schedule is feasible for the OR-instance with precedence graph G', it is also feasible for the initial precedence graph G. By Lemma 4, there exists an optimal solution with makespan $C^*_{\max}$ for the instance on G that is also feasible for the instance on G'. Since the schedule returned by ALGOPMTN is optimal for the instance on G', it holds $C_{\max} \leq C^*_{\max}$. This proves the claim.

As for the running time, note that we can construct an earliest start schedule in time $O(n)$, since we have to consider each job exactly once. The same is true for constructing the closed collection of minimal chains and the graph G'. Finally, we apply Lawler's algorithm, which has a running time of $O(n^2)$ (Lawler 1982). Therefore, we obtain a total running

time of $O(n^2)$ for arbitrary processing times. In case of unit processing times, can apply the algorithm of Monma (1982), which runs in time $O(n)$, instead of the algorithm of Lawler (1982). Hence, the running time of ALGOPMTN reduces to $O(n)$ if $p_j = 1$ for all $j \in [n]$. □

5 Concluding remarks

In this paper, we discuss the problem of minimizing the makespan on parallel identical machines with OR-precedence constraints. We introduce the concept of minimal chains, and use it to prove that List Scheduling (Graham 1966) achieves an approximation guarantee of 2. Further, we prove that it is NP-hard to obtain an approximation factor strictly better than $4/3$ via a reduction from VERTEX COVER. Using minimal chains, we show that there exists an optimal preemptive schedule of a certain structure and exploit this structure to obtain a polynomial-time algorithm for the preemptive variant. The running time of our algorithm is $O(n^2)$ for arbitrary processing times and $O(n)$ for unit processing times.

The results presented here match the complexity and best-known approximation guarantees of makespan minimization if the precedence graph is an outtree, which is a special case where AND- and OR-precedence constraints coincide. Clearly, any improvement on OR-precedence constraints directly transfers to AND-precedence constraints on outtrees. On the other hand, due to the close connection with minimal chains, any progress on the approximation factor of AND-precedence constraints on outtrees might also be applicable to OR-precedence constraints.

Note that NP-hardness of obtaining a $4/3 - \varepsilon$ approximation coincides with the result of Lenstra and Rinnooy Kan (1978) for the corresponding problem with AND-precedence constraints. For the latter problem, there is a conditional lower bound of 2 on the approximation factor under a variant of the Unique Games Conjecture (Khot 2002; Bansal and Khot 2009; Svensson 2010). It would be interesting to obtain a similar (conditional) lower bound for OR-precedence constraints.

Acknowledgements The author thanks the anonymous referees and associate editor for helpful and valuable comments on improving the presentation. This work has been supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF).

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Bansal, N., & Khot, S. (2009). Optimal long code test with one free bit. In *Proceedings of the 50th annual IEEE symposium on foundations of computer science*. (pp. 453–462). IEEE.

Brucker, P., Garey, M. R., & Johnson, D. S. (1977). Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Mathematics of Operations Research*, *2*(3), 275–284.

Chen, B., & Vestjens, A. P. A. (1997). Scheduling on identical machines: How good is LPT in an on-line setting? *Operations Research Letters*, *21*(4), 165–169.

Du, J., Leung, J. Y. T., & Young, G. H. (1991). Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, *92*(2), 219–236.

Erlebach, T., Kääb, V.,& Möhring, R. H. (2003). Scheduling AND/OR-networks on identical parallel machines. In *International workshop on approximation and online algorithms, no. 2909 in LNCS*. (pp. 123–136). Springer.

Garey, M. R., & Johnson, D. S. (1978). Strong NP-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, *25*(3), 499–508.

Gillies, D. W., & Liu, J. W. S. (1995). Scheduling tasks with AND/OR precedence constraints. *SIAM Journal on Computing*, *24*(4), 797–810.

Gonzalez, T. F., & Johnson, D. B. (1980). A new algorithm for preemptive scheduling of trees. *Journal of the ACM (JACM)*, *27*(2), 287–312.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, *45*(9), 1563–1581.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, *17*(2), 416–429.

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. In *Annals of discrete mathematics*. (Vol. 5, pp. 287–326). Elsevier.

Hall, L. A., & Shmoys, D. B. (1989). Approximation schemes for constrained scheduling problems. In *Proceedings of the 30th annual symposium on foundations of computer science*. (pp. 134–139). IEEE.

Hochbaum, D. S., & Shmoys, D. B. (1988). A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, *17*(3), 539–551.

Hu, T. C. (1961). Parallel sequencing and assembly line problems. *Operations Research*, *9*(6), 841–848.

Jansen, K. (2010). An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, *24*(2), 457–485.

Johannes, B. (2005). On the complexity of scheduling unit-time jobs with OR-precedence constraints. *Operations Research Letters*, *33*(6), 587–596.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer.

Khot, S. (2002). On the power of unique 2-prover 1-round games. In *Proceedings of the 34th annual ACM symposium on theory of computing*. (pp. 767–775). ACM.

Lawler, E. L. (1982). Preemptive scheduling of precedence-constrained jobs on parallel machines. In *Deterministic and stochastic scheduling*. (pp. 101–123). Springer.

Lenstra, J. K., & Rinnooy Kan, A. H. G. (1978). Complexity of scheduling under precedence constraints. *Operations Research*, *26*(1), 22–35.

Lenstra, J. K., Rinnooy Kan, A. H. G.,& Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of discrete mathematics*. (Vol. 1, pp. 343–362). Elsevier.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, *6*(1), 1–12.

Mnich, M., & Wiese, A. (2015). Scheduling and fixed-parameter tractability. *Mathematical Programming*, *154*(1–2), 533–562.

Möhring, R. H., Skutella, M., & Stork, F. (2004). Scheduling with AND/OR precedence constraints. *SIAM Journal on Computing*, *33*(2), 393–415.

Monma, C. L. (1982). Linear-time algorithms for scheduling on parallel processors. *Operations Research*, *30*(1), 116–124.

Muntz, R. R.,& Coffman Jr, E. G. (1970). Preemptive scheduling of real-time tasks on multiprocessor systems. *Journal of the ACM (JACM), 17*(2), 324–338.

Svensson, O. (2010). Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the 42nd annual ACM symposium on theory of computing*. (pp. 745–754). ACM.

Ullman, J. D. (1975). NP-complete scheduling problems. *Journal of Computer and System Sciences*, *10*(3), 384–393.