Parameterized Complexity of Scheduling Unit-Time Jobs with Generalized Precedence Constraints

Christina Büsing
Research and Teaching Area for Combinatorial Optimization, RWTH Aachen University, Germany

Maurice Draeger
RWTH Aachen University, Germany

Corinna Mathwieser[1]
Research and Teaching Area for Combinatorial Optimization, RWTH Aachen University, Germany

Abstract

We study the parameterized complexity of scheduling unit-time jobs on parallel, identical machines under generalized precedence constraints for minimization of the makespan and the sum of completion times ($P|gen\text{-}prec,\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j\}$). In our setting, each job is equipped with a Boolean formula (precedence constraint) over the set of jobs. A schedule satisfies a job's precedence constraint if setting earlier jobs to true satisfies the formula. Our definition generalizes several common types of precedence constraints: classical *and*-constraints if every formula is a conjunction, *or*-constraints if every formula is a disjunction, and *and*/*or*-constraints if every formula is in conjunctive normal form. We prove fixed-parameter tractability when parameterizing by the number of predecessors. For parameterization by the number of successors, however, the complexity depends on the structure of the precedence constraints. If every constraint is a conjunction or a disjunction, we prove the problem to be fixed-parameter tractable. For constraints in disjunctive normal form, we prove $W[1]$-hardness. We show that the *and*/*or*-constrained problem is $\mathcal{NP}$-hard, even for a single successor. Moreover, we prove $\mathcal{NP}$-hardness on two machines if every constraint is a conjunction or a disjunction. This result not only proves para-$\mathcal{NP}$-hardness for parameterization by the number of machines but also complements the polynomial-time solvability on two machines if every constraint is a conjunction ([4]) or if every constraint is a disjunction ([11]).

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases scheduling, precedence constraints, fixed-parameter tractability, complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.7

1 Introduction

Scheduling jobs with precedence constraints on identical parallel machines is a well-researched problem with many real-world applications. A classical precedence constraint between two jobs, say from job i to job j, means that job j cannot begin until job i has finished. If job j has multiple such constraints, it can only start once all the predecessor jobs have been completed. However, many real-life settings find classical precedence constraints (*and*-type constraints) too restrictive. It may not always be necessary to complete all predecessors before starting a job. This need for flexibility has led to the proposal of alternative variants of precedence constraints in the literature. The first variant is *or*-precedence constraints where completing any predecessor is sufficient to start a job. The second variant includes *and+or*-constraints where some jobs require all predecessors to be completed earlier (*and*-jobs) while others only require to be preceded by only one predecessor (*or*-jobs). Another more general variant is known as *and*/*or*-constraints. In this model, each job has multiple groups of predecessor jobs, and to start the job, at least one job from each group must be completed first. A motivating

[1] corresponding author

example for *and/or*-constraints is provided by Goldwasser, Lotombe and Motwani ([7]), who examine the problem of disassembling a product. Certain parts may need to be taken out prior to accessing other components. Occasionally, the same part might be accessible from various directions and therefore, removing any of several different components may provide access to a particular part. Scheduling with a combination of *or*- and *and*-constraints has further been studied in the works of Gillies and Liu [6], Möhring et al. [17], Lee et al. [13] and Erlebach et al. [5]. For scheduling under solely *or*-constraints, we refer to the works of Berit [11] and the paper by Happach [10].

Given the diverse landscape of precedence constraint types, we propose a more general notion of precedence constraints, which encompasses all aforementioned types. Specifically, we encode precedence constraints regarding a specific job j as a logical formula ψ_j over (unnegated) variables representing the set J of jobs: $\{x_i \mid i \in J\}$. In any feasible schedule, ψ_j must be satisfied by the assignment resulting from setting $x_i = 1$ precisely if job i is executed prior to j. Our definition gives rise to *and*-constraints when each ψ_j is a conjunction; to *or*-constraints when each ψ_j is a disjunction; to *and+or*-constraints when each ψ_j is either; and to *and/or*-constraints when each ψ_j takes conjunctive normal form. Naturally, another interesting variant arises when each ψ_j is expressed in disjunctive normal form. We refer to this new variant as *or/and*-constraints. To gain an intuitive motivation of *or/and*-constraints, consider the following example from disaster management: There are several routes that can be used to evacuate people from a village, but obstacles (such as debris or fallen trees) are blocking these routes. To proceed with the evacuation, it might not be essential to clear all routes yet but it is necessary to remove all obstacles from one route.

The study of parameterized algorithms as an approach to addressing computationally hard problems is a vibrant field of research that has gained significant momentum in the scheduling community in recent years ([15], [19], [3], [1], [18], [16], [12]). In the context of classical precedence-constrained scheduling problems, the most common parameters include the number of machines, the width of the partial order that is induced by the precedence constraints, and the maximum processing time. Observe that the effect of precedence constraints on the complexity of scheduling problems can be twofold: On the one hand, the additional structure imposed by precedence constraints can simplify scheduling problems. An extreme example is when precedence constraints result in a total order among all jobs, rendering scheduling problems trivial. On the other hand, precedence constraints can also turn polynomial-time solvable scheduling problems into $\mathcal{NP}$-hard problems (Prot and Bellenguez-Morineau [19]): Scheduling unit-time jobs on parallel identical machines while minimizing the makespan or the (weighted) sum of completion times becomes $\mathcal{NP}$-hard in the presence of *and*-precedence constraints (Lenstra and Kan [14], Sethi [20]). Restricting the width of the partial order in classical precedence-constrained scheduling seeks to exploit the former effect: A small width generates high interrelatedness between the jobs and may thus provide scheduling problems with more structure. Yet, this approach does not necessarily result in fixed-parameter tractability, not even in combination with other classical parameters. Bodlaender and Fellows ([2]) show that even $P|prec,\ p_j = 1|C_{\max}$ is $W[2]$-hard when parameterized by the width and the number of machines and Van Bevern et al. show in [21] that $P2|prec,\ p_j \in \{1, 2\}|C_{\max}$ is $W[2]$-hard when parameterized by the width. The diversity in the forms that precedence constraints can take – and the observation that enforcing dense dependency structures does not necessarily improve tractability – motivates the introduction of new parameters for generalized precedence constraints which limit interrelatedness.

Our Contribution

We study the problem $P|gen\text{-}prec,\ p_j = 1|\gamma$ of scheduling unit-time jobs on parallel, identical machines under generalized precedence constraints and consider the objectives of minimizing the makespan ($\gamma = C_{\max}$) or the sum of completion times ($\gamma = \sum_j C_j$). We investigate the parameterized complexity with respect to three parameters: the number k_p of predecessors, the number k_s of successors and the number m of machines. We achieve fixed-parameter tractability when parameterizing by the total number k_p of jobs that appear as predecessors in any precedence constraint, by providing a single-exponential algorithm. For the number k_s of successors however, the picture is more diverse and depends on the structure of the precedence constraints. For *and+or*-constraints, we prove the problem to be fixed-parameter tractable (FPT) by providing an algorithm with doubly-exponential dependence on k_s. This result does not extend to *and/or*- nor to *or/and*-constraints: We prove that $P|and/or\text{-}prec,\ p_j = 1|\gamma$ is para-$\mathcal{NP}$-hard with respect to k_s by showing that the problem is $\mathcal{NP}$-hard, even for a single successor. For *or/and*-constraints, we prove the problem to be in XP and to be $W[1]$-hard. Since Berit provides in [11] a polynomial-time algorithm for scheduling unit-time jobs on parallel, identical machines under *or*-constraints, it makes sense to ask if *and+or*-constrained scheduling remains FPT when only restricting the number $k_s^{and} \leq k_s$ of *and*-successors. However, we prove that scheduling under *and+or*-constraints is $\mathcal{NP}$-hard, even if there is only one *and*-job. Since our reduction only makes use of two machines, we also obtain that $P|and{+}or,\ p_j = 1|\gamma$ is para-$\mathcal{NP}$-hard with respect to the number m of machines.

2 Preliminaries

Problem Definition

We consider the problem of non-preemptively scheduling unit time jobs on identical, parallel machines under generalized precedence constraints. An instance specifies a set $J = \{1, \dots, n\}$ of jobs, a number m of identical machines and occasionally job weights $w_j \geq 0$, $j \in J$. A schedule $\mathcal{S}$ specifies which job, if any, each machine works on at each point in time. More precisely, a schedule $\mathcal{S}$ is represented as tuple $(C_j)_{j \in J}$, where $C_j \in \mathbb{N}$ indicates the *completion time* of each job $j \in J$. Since m jobs can be processed in parallel, we assume that $|\{j \in J \mid C_j = t\}| \leq m$ for any time $t \in \mathbb{N}$. We say that a job i *precedes* a job j in $\mathcal{S}$ if $C_i < C_j$. Moreover, we refer to the period between time $t-1$ and time t as *time slot* $t \in \mathbb{N}$.

Precedence constraints comprehend the requirement that certain jobs have to be finished before others can be started. In order to formalize precedence constraints, we introduce a Boolean variable $x_j \in \{0,1\}$ for each job $j \in J$. Moreover, we denote by $\mathbb{1}$ the logical one that represents the truth value "true". We encode precedence constraints as family $(\psi_j)_{j \in J}$, where each ψ_j, $j \in J$, is a Boolean formula over the (unnegated) variables in $\{x_i \mid i \in J\} \cup \{\mathbb{1}\}$. If a variable x_i occurs in a precedence constraint ψ_j, $i, j \in J$, then i is called a *predecessor of* j and j is called a *successor of* i. A job $i \in J$, which is the predecessor of some other job, is called a *predecessor*. Likewise, a job $j \in J$ with non-trivial precedence constraint $\psi_j \neq \mathbb{1}$, is called a *successor*. We refer by k_p (k_s) to the number of predecessors (successors).

A schedule $\mathcal{S}$ is associated with variable assignments $x^j(\mathcal{S})$, $j \in J$, which set $x_i^j(\mathcal{S}) = 1$ precisely if i precedes j in $\mathcal{S}$. A schedule $\mathcal{S}$ is called *feasible* with respect to precedence constraints $(\psi_j)_{j \in J}$ if each ψ_j, $j \in J$, is satisfied by setting $x_i = x_i^j(\mathcal{S})$ for $i \in J$. Given a schedule $\mathcal{S}$, we call a job j *available* at time t if the variable assignment that sets $x_i = 1$ precisely if $C_i \leq t$ is a true assignment for ψ_j. We consider three classical objectives: Minimizing the makespan ($\gamma := C_{\max} = \max_{j \in J} C_j$), minimizing the sum of completion times

($\gamma := \sum_j C_j$) or minimizing the weighted sum of completion times, ($\gamma := \sum_j w_j C_j$). Following standard notation as introduced by Graham et al. in [9], we refer to the problem of finding an optimal schedule by $P|\mathit{gen\text{-}prec},\ p_j = 1|\gamma$. Here, P means identical parallel machines, $p_j = 1$ indicates unit processing times for all jobs, and $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ specifies the objective function. If we consider the problem for two instead of arbitrarily many machines, we write $P2|\mathit{gen\text{-}prec},\ p_j = 1|\gamma$. Given an instance I of $P|\mathit{gen\text{-}prec},\ p_j = 1|\gamma$, we denote the objective value of an optimal solution by $OPT(I)$ and of a specific schedule $\mathcal{S}$ by $\gamma(\mathcal{S})$. We distinguish five special cases of $P|\mathit{gen\text{-}prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$:

- $P|\mathit{prec},\ p_j = 1|\gamma$, where each ψ_j is a conjunction,
- $P|\mathit{or\text{-}prec},\ p_j = 1|\gamma$, where each ψ_j is a disjunction,
- $P|\mathit{and{+}or\text{-}prec},\ p_j = 1|\gamma$, where each ψ_j is either a conjunction or a disjunction,
- $P|\mathit{and/or\text{-}prec},\ p_j = 1|\gamma$, where each ψ_j is in conjunctive normal form,
- $P|\mathit{or/and\text{-}prec},\ p_j = 1|\gamma$, where each ψ_j is in disjunctive normal form.

Feasibility and objectives

A feasible schedule for $P|\mathit{gen\text{-}prec},\ p_j = 1|\gamma$ can be calculated in polynomial time by repeatedly choosing an available job and assigning it to any free machine. For a precise description, we refer to Algorithm 1 by Möhring et al. [17], which was formulated for *and/or*-constraints but can be easily implemented for generalized precedence constraints. In case of classical *and*-constraints, infeasibility is equivalent to the existence of a cycle in the precedence graph G, where G is a digraph with node set J and arc set

$$A = \{(i, j) \mid i, j \in J,\ i \text{ is a predecessor of } j\}.$$

Throughout this paper, we assume that all instances are feasible.

Scheduling with unit-time jobs has the advantage that we can minimize the makespan and the sum of completion times simultaneously. While not every feasible schedule with minimum makespan necessarily optimizes the sum of completion times, the reverse holds true (which can be seen through standard swapping arguments). Subsequently, it suffices to only prove the correctness of algorithms for $\gamma = \sum_j (w_j) C_j$ and hardness for $\gamma = C_{\max}$.

List Scheduling

List scheduling is a simple and intuitive method for addressing scheduling problems and was introduced by Graham in [8]. The approach involves assigning a priority to each job, creating an ordered list based on these priorities. Jobs are then scheduled in the order they appear on this list. Whenever a machine is free, it is assigned the first unscheduled job from the list that is available to be processed at that time. List scheduling ensures machines are never left idle intentionally – idle time only occurs when no jobs are currently available for processing. This method can be efficiently implemented and runs in polynomial time.

3 Parameterization by the number of predecessors

In this section, we analyze the parameterized complexity regarding the overall number k_p of predecessors. Note that the parameters k_p and k_s are, in a sense, conceptually opposed to width. While a small width typically indicates that most jobs are highly interrelated, having few predecessors or successors often leads to many jobs being largely independent. A small width is therefore used to enforce structure in scheduling problems that are hard without precedence constraints. In contrast, parameterizing by k_p or k_s aims to capture the effect of introducing a limited number of precedence constraints into problems that are efficiently solvable when no precedence constraints are present (as is the case for unit processing times). In the following, we prove that $P|\textit{gen-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is FPT when parameterized by the number of predecessors (see Theorem 2). For minimizing the weighted sum of completion times, we prove the weaker result that $P|\textit{gen-prec},\ p_j = 1|\sum_j w_j C_j$ lies in $\mathcal{P}$ if k_p is bounded by a constant. In either case, we provide an algorithm that essentially explores all possibilities of positioning the predecessors in a schedule and later inserts all other jobs via List Scheduling. To test fewer possibilities, we argue that we may assume all predecessors to be processed before time step k_p if $\gamma \in \{C_{\max}, \sum_j C_j\}$.

▶ **Lemma 1.** *Let $I = (J, (\psi_j)_{j\in J}, m)$ be a feasible instance of $P|\textit{gen-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ with a set $J_p \subseteq J$ of k_p predecessors. Then there exists an optimal schedule $\mathcal{S}^*$ with completion times $(C_j^*)_{j\in J}$ such that $C_j^* \leq k_p$ for all $j \in J_p$.*

Proof. Let $\mathcal{S}^* = (C_j^*)_{j\in J}$ be a schedule with optimal sum of completion times. Assume $C_j^* > k_p$ for some $j \in J_p$. Let $t^* \leq k_p$ be the earliest time slot where no predecessor is processed. Choose a job $i \in J$ with $C_i^* = t^*$. By choice of t^*, i is not a predecessor job. Let h be an earliest predecessor with $C_h^* > t^*$. Construct a schedule $\mathcal{S}$ from $\mathcal{S}^*$ by swapping i and h. Clearly, the swap is feasible and does not alter the sum of completion times. Thus $\mathcal{S}$ is optimal too, while the earliest time slot t where no predecessor is processed has increased, $t > t^*$. Repeat until all completion times of predecessor jobs do not exceed k_p. ◀

The algorithm ALG-PRE for instances of $P|\textit{gen-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$, with predecessors $J_p = \{i_1, \dots, i_{k_p}\} \subseteq J$ proceeds as follows: Repeatedly pick a configuration $(C_{i_\ell})_{\ell=1}^{k_p}$ of predecessor completion times from $\mathcal{C} := \{1, \dots, k_p\}^{k_p}$ if $\gamma \in \{C_{\max}, \sum_j C_j\}$ or from $\mathcal{C} := \{1, \dots, n\}^{k_p}$ if $\gamma = \sum_j w_j C_j$. If $(C_{i_\ell})_{\ell=1}^{k_p}$ is a feasible schedule for J_p, insert all remaining jobs through List Scheduling in order of non-increasing weight and return the schedule with the smallest objective value.

▶ **Theorem 2.** *Let I be a feasible instance of $P|\textit{gen-prec},\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ with k_p predecessors. If $\gamma \in \{C_{\max}, \sum_j C_j\}$, then ALG-PRE returns in $\mathcal{O}(k_p^{k_p} \cdot m \cdot n^2)$ time the completion times of an optimal schedule $\mathcal{S}^*$. Otherwise, ALG-PRE returns in $\mathcal{O}(n^{k_p+2} \cdot m)$ time the completion times of an optimal schedule $\mathcal{S}^*$.*

Proof. Let $\mathcal{S}^* = (C_j^*)_{j\in J}$ be an optimal schedule, such that $(C_{i_1}^*, \dots, C_{i_{k_p}}^*) \in \mathcal{C}$. (The existence of such $\mathcal{S}^*$ follows from Lemma 1). Consider the iteration of ALG-PRE where $C_j = C_j^*$ for all $j \in J_p$. Clearly, the algorithm computes a feasible schedule $\mathcal{S} = (C_j)_{j\in J}$. Denote by k_p^t the number of predecessor jobs processed during time slot $t \in \{1, \dots, n\}$.

We prove the optimality of $\mathcal{S} = (C_j)_{j\in J}$ for $\gamma = \sum_j w_j C_j$, from which the statement for the other two objectives follows immediately. Assume the jobs in $J \setminus J_p$ have $1 \leq q \leq n - k_p$ different weights that we denote by $w^1 \geq w^2 \geq \ldots \geq w^q$. We consider the vector $z(t) = (z_1(t), \dots, z_q(t)) \in \mathbb{N}^q$ where $z_i(t)$, $i \in \{1, \dots, q\}$, denotes the number of jobs in $J \setminus J_p$ with weight w^i that $\mathcal{S}$ processes during time slot $t \in \mathbb{N}$. For $\mathcal{S}^*$, we define $z^*(t)$, $t \in \mathbb{N}$, analogously. If $z(t) = z^*(t)$ for all $t \in \mathbb{N}$, then $\sum_j w_j C_j = \sum_j w_j C_j^*$ and $\mathcal{S}$ is optimal. Otherwise, let $T := \min\{t \mid z(t) \neq z^*(t)\}$ be the smallest time slot for which $z(T), z^*(T)$ are not equal.

Assume first that $z(T)$ is lexicographically smaller than $z^*(T)$. Let $i \in \{1, \dots, q\}$ be such that $z_s^*(T) = z_s(T)$ for $s < i$ and $z_i^*(T) > z_i(T)$. It holds that $\sum_{t=1}^{T} z_i(t) < \sum_{t=1}^{T} z_i^*(t)$. Subsequently, there exists a job $j \in J$ with weight $w_j = w^i$, such that $C_j^* \leq T$ but $C_j > T$. Observe that j is not a predecessor, otherwise $C_j = C_j^*$. Since $C_j^* \leq T$, job j is available at time $T-1$ in $\mathcal{S}^*$ and thus also in $\mathcal{S}$. Since ALG-PRE sets $C_j > T$, the following holds: When j is considered, the algorithm has already occupied all machines during time slot T, namely with predecessor jobs or jobs with weight at least $w_j = w^i$. This yields a contradiction as

$$\begin{aligned} m = k_p^T + \sum_s z_s(T) = k_p^T + \sum_{s<i} z_s(T) + z_i(T) = k_p^T + \sum_{s<i} z_s^*(T) + z_i(T) \\ < k_p^T + \sum_{s<i} z_s^*(T) + z_i^*(T) \le m. \end{aligned}$$

Assume now that $z^*(T)$ is lexicographically smaller than $z(T)$ and let again $i \in \{1, \dots, q\}$ be such that $z_s^*(T) = z_s(T)$ für $s < i$ while $z_i^*(T) < z_i(T)$. By analogous arguments, there exists a job $j \in J \setminus J_p$ with weight w^i such that $C_j \le T$, $C_j^* > T$ and j is available at time $T-1$ in $\mathcal{S}^*$.

The optimal schedule $\mathcal{S}^*$ occupies all machines during time slot T with predecessor jobs or jobs of weight at least $w_j = w^i$: Otherwise, swapping j with a job of smaller weight or placing it on an idle machine during time slot T would yield a feasible solution with smaller weighted sum of completion times. As before,

$$m = k_p^T + \sum_{s<i} z_s^*(T) + z_i^*(T) < k_p^T + \sum_{s<i} z_s(T) + z_i(T) \le m$$

yields a contradiction.

We conclude with a brief runtime analysis: We sort all non-predecessor jobs with respect to weight once ($\mathcal{O}(n \log n)$ time). Then the algorithm checks $|\mathcal{C}|$ configurations $(t_1, \dots, t_{k_p}) \in \mathcal{C}$ of predecessor completion times. For each such configuration, the algorithm inserts $\mathcal{O}(n)$ jobs $j \in J \setminus J_p$. For each job $j \in J \setminus J_p$, it checks for $\mathcal{O}(n)$ time steps t whether j is available at time t. For any such time step t, the algorithm searches for an idle machine among m machines. Since, $|\mathcal{C}| \in \mathcal{O}(k^k)$ if $\gamma \in \{C_{\max}, \sum_j C_j\}$ and $|\mathcal{C}| \in \mathcal{O}(n^k)$ otherwise, the claimed asymptotic runtime follows. ◀

4 Parameterization by the number of successors

We investigate whether restricting the number of successors has the same effect as restricting the number of predecessors. However, it turns out that the parameterized complexity with respect to the number of successors depends on the structure of the precedence constraints. More precisely, we prove that $P|$*and+or-prec*$,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is FPT when parameterized with respect to the number k_s of successors. For the more general *or/and*-constraints, we prove that $P|$*or/and-prec*$,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is polynomial-time solvable if the number k_s of successors is bounded by a constant. We show that this result cannot be improved to fixed-parameter tractability by proving that $P|$*or/and-prec*$,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is $W[1]$-hard when parameterized by the number k_s of successors. For *and/or*-constraints, we prove para-$\mathcal{NP}$-hardness. Throughout this section, we assume $\gamma \in \{C_{\max}, \sum_j C_j\}$.

To prove that $P|$*and+or-prec*$,\ p_j = 1|\gamma$ is FPT with respect to k_s, we proceed as follows: 1. We provide an FPT algorithm AND-ALG-SUCC for instances of $P|$*prec*$,\ p_j = 1|\gamma$ with k_s successors. 2. We argue how to solve $P|$*and+or-prec*$,\ p_j = 1|\gamma$ by repeatedly calling AND-ALG-SUCC. Algorithm AND-ALG-SUCC is similar to ALG-PRE and explores all possibilities of positioning the successors in a schedule before inserting all other jobs. To avoid naïvely testing all $\mathcal{O}(n^{k_p})$ possible ways to schedule the successors, we use a similar result to Lemma 1: We may assume all successors to be processed within a time span of length k_s.

▶ **Lemma 3.** *Let $I = (J, (\psi_j)_{j \in J}, m)$ be a feasible instance of $P|$gen-prec, $p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ with k_s successors and let $J_s \subseteq J$ denote the set of successors. Then there exists an optimal schedule $\mathcal{S}^* = (C_j^*)_{j \in J}$, such that $C_j^* - C_i^* \leq k_s - 1$ for all $i, j \in J_s$.*

Proof. Consider an optimal schedule $\mathcal{S}^*$ with completion times $\bar{C}_\ell$, $\ell \in J$, and assume that $\bar{C}^s_{\max} - \bar{C}^s_{\min} > k_s - 1$ where $\bar{C}^s_{\min} = \min\{\bar{C}_\ell \mid \ell \in J_s\}$ is the completion time of an earliest successor and $\bar{C}^s_{\max} = \max\{\bar{C}_\ell \mid \ell \in J_s\}$ is the completion time of a latest successor. Let t^* be the earliest time slot with $\bar{C}^s_{\min} < t^* < \bar{C}^s_{\max}$ where no successor is processed. Pick a job $j \in J \setminus J_s$ with $C_j^* = t^*$. Let i be a successor with $C_i^* = t^* - 1$. We construct from $\mathcal{S}^*$ another feasible schedule $\mathcal{S}$ with completion times C_ℓ, $\ell \in J$, by swapping jobs i and j. Clearly, $\mathcal{S}$ is feasible and optimal and $C^s_{\max} := \max\{C_\ell \mid \ell \in J_s\} = \bar{C}^s_{\max}$. If $t^* - 1 = \bar{C}^s_{\min}$ and i is the only successor with $\bar{C}_i = t^* - 1$, then $C^s_{\min} := \min\{C_\ell \mid \ell \in J_s\} > \bar{C}^s_{\min}$. Else, if $t^* - 1 > \bar{C}^s_{\min}$ and i is again the only successor with $\bar{C}_i = t^* - 1$, then $C^s_{\min} = \bar{C}^s_{\min}$ but the earliest time slot $t \in (C^s_{\min}, C^s_{\max})$ without successors in $\mathcal{S}$ has decreased, $t < t^*$. Otherwise, we also get $C^s_{\min} = \bar{C}^s_{\min}$ while the overall number of time slots $t \in (C^s_{\min}, C^s_{\max})$ without successors in $\mathcal{S}$ has decreased. By repeating the argument, we obtain an optimal solution where all successors are processed within a time span of length k_s. ◀

Given a set $J_s \subseteq J$ of successors, the algorithm AND-ALG-SUCC repeatedly picks a configuration $(C_j)_{j \in J_s}$ from

$$\mathcal{C} := \{(t_j)_{j \in J_s} \in \{1, \ldots, n\}^{k_s} \mid \forall i, j \in J_s : t_i - t_j \leq k_s - 1\},$$

such that $|\{j \in J_s \mid C_j = t\}| \leq m$ for all $t \in \{1, \ldots, n\}$ and $t_i < t_j$ if i is a predecessor of j. Then the algorithm inserts the remaining jobs as follows: AND-ALG-SUCC iterates over all successors in order of non-decreasing completion times. For each successor $j \in J_s$, the algorithm assigns the earliest possible completion time to each predecessor. If some predecessor cannot be scheduled before j, the respective schedule of successors is discarded. In the end, all remaining jobs (which are neither successor nor predecessor) are inserted as early as possible.

▶ **Proposition 4.** *Let I be a feasible instance of $P|prec$, $p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ with k_s successors. Then* AND-ALG-SUCC *returns in $\mathcal{O}(k_s^{k_s} \cdot m \cdot n^3)$ time an optimal schedule $\mathcal{S}^*$.*

Proof. By Lemma 3, there exists an optimal schedule $\mathcal{S}^*$ with completion times C_j^*, $j \in J$ such that $(C^*_{i_1}, \ldots, C^*_{i_{k_s}}) \in \mathcal{C}$. Consider the iteration of AND-ALG-SUCC where $C_j = C_j^*$ for all $j \in J_s$. Since $\mathcal{S}^*$ is a feasible schedule with identical successor completion times, there is enough space on the machines prior to each successor $j \in J_s$ for AND-ALG-SUCC to schedule all of j's predecessors. Hence, $(C^*_{i_1}, \ldots, C^*_{i_{k_s}})$ is not discarded and AND-ALG-SUCC computes a feasible schedule $\mathcal{S}$. We proceed by proving that $\mathcal{S}$ is in fact optimal. We define $z(t)$ as the number of jobs in $\{j \in J \setminus J_s \mid C_j = t\}$, which $\mathcal{S}$ processes during time slot $t \in \mathbb{N}$. For $\mathcal{S}^*$, we define $z^*(t)$, $t \in \mathbb{N}$, analogously. If $z(t) = z^*(t)$ for all $t \in \mathbb{N}$, then $(C_{\max}, \sum_j C_j) = (C^*_{\max}, \sum_j C^*_j)$ and $\mathcal{S}$ is optimal. Otherwise, let $T := \min\{t \mid z(t) \neq z^*(t)\}$ be the smallest time slot for which $z(T), z^*(T)$ are not equal.

Assume first that $z(T) < z^*(T)$. It holds that $\sum_{t=1}^T z(t) < \sum_{t=1}^T z^*(t)$. Subsequently, there exists a job $j \in J$, such that $C_j^* \leq T$ but $C_j > T$. Observe that j is no successor, otherwise $C_j = C_j^*$. When AND-ALG-SUCC assigns a completion time to j, there is an idle machine during time slot T because $z(T) < z^*(T) \leq m$ and AND-ALG-SUCC assigns a completion time $C_j \leq T$ to j, contradiction. If $z(T) > z^*(T)$, there exists a job $j \in J \setminus J_s$, such that $C_j^* > T$ but $C_j \leq T$. The optimal schedule $\mathcal{S}^*$ has an idle machine during time slot T because $z^*(T) < z(T) \leq m$. Adapting the optimal solution such that $C_j^* = T$ does not

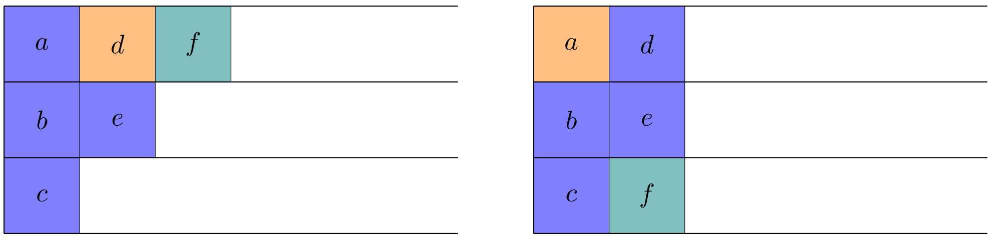


Figure 1 We process six jobs a, b, c, d, e, f with *and+or*-precedence constraints $\psi_e = x_a \wedge x_b \wedge x_c$, $\psi_f = x_a \vee x_d$ on three machines. The left schedule is optimal for the *and*-variation with $\psi'_f = x_d$. The schedule on the right is optimal for the *and*-variation with $\psi'_f = x_a$ and for the original instance.

negatively affect neither the optimality of $\mathcal{S}^*$ nor its feasibility since j has no predecessors. We repeat this argument until $z(t) = z^*(t)$ for all $t \in \mathbb{N}$, which proves that $\mathcal{S}$ is optimal. We omit a runtime analysis since it is analogous to the proof of Theorem 2. ◀

Using AND-ALG-SUCC, it is easy to see that $P|or/and\text{-}prec,\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j\}$, lies in $\mathcal{P}$ if the number of successors $k_s \leq const$ is bounded by some constant $const \geq 0$: Recall that *or/and*-constraints require each precedence constraint ψ_j, $j \in J$, to be a disjunction of conjunctive clauses. By electing one clause per job, we can solve an instance I of $P|or/and\text{-}prec,\ p_j = 1|\gamma$ by solving $\mathcal{O}(|I|^{k_s})$ instances of $P|prec,\ p_j = 1|\gamma$ with at most k_s successors. We state this in the following corollary:

▶ **Corollary 5.** *$P|or/and\text{-}prec,\ p_j = 1|\gamma$ can be solved in polynomial-time if the number of successors $k_s \leq const$ is bounded by a constant $const \geq 0$.*

For *and+or*-constraints, we can strengthen this argument to prove that $P|and{+}or\text{-}prec,\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j\}$ is FPT. Consider an instance $I = (J, (\psi_j)_{j \in J}, m)$ of $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ with k_s successors. Let $J_s = J_s^{and} \dot\cup J_s^{or}$ be a partition of the set of successors, where a precedence constraint ψ_j of a job $j \in J_s$ is a conjunction if $j \in J_s^{and}$ and a disjunction if $j \in J_s^{or}$. We call jobs in J_s^{and} *and*-jobs and jobs in J_s^{or} *or*-jobs. Consider an *or*-job $j \in J_s^{or}$ and let $i \in J$ be a predecessor of j. Consider the alternate instance $I' = (J, (\psi'_j)_{j \in J}, m)$ that we obtain by fixing i as only predecessor of j, that is, $\psi'_\ell = \psi_\ell$ if $\ell \neq j$ and $\psi'_j = x_i$. Clearly, any optimal schedule $\mathcal{S}'$ for I' is feasible for I too. However, whether $\mathcal{S}'$ is optimal for I too, strongly depends on whether i is also a predecessor of other jobs in $J_s \setminus \{j\}$. See Figure 1 for an example. Consider an arbitrary feasible schedule $\mathcal{S}$ for I. For each *or*-job $j \in J_s^{or}$, we can determine a predecessor job i_j of j such that i_j precedes j in $\mathcal{S}$. Observe that we can possibly have $i_j = i_k$ for two different *or*-jobs $j, k \in J_s^{or}$, $j \neq k$, and that each i_j may be a predecessor of multiple *and*-successors. In other words, the choice of $(i_j)_{j \in J_s^{or}}$ gives rise to a function $\sigma : J_s^{or} \to 2^{J_s}$ where $k \in \sigma(j)$ precisely if k is an *or*-job and $i_j = i_k$ or if k is an *and*-job with predecessor i_j. Our algorithm essentially tests all relevant variations of σ and picks an *and*-predecessor $i_j \in J$ for each *or*-job $j \in J_s$ according to σ, before it applies AND-ALG-SUCC to the resulting instance with *and*-constraints.

▶ **Definition 6.** *Let $I = (J, (\psi_j)_{j \in J}, m)$ be an instance of $P|and{+}or\text{-}prec,\ p_j = 1|\gamma,\ \gamma \in \{C_{\max}, \sum_j C_j\}$ where $J_s = J_s^{or} \dot\cup J_s^{and}$ is the set of successors. For each or-job $j \in J_s^{or}$, let $i_j \in J$ be a predecessor of j.*

1. *Let $\sigma : J_s^{or} \to 2^{J_s}$ be a function such that $k \in \sigma(j)$, $j \in J_s^{or}$, precisely if $k \in J_s^{or}$ and $i_j = i_k$ or if $k \in J_s^{and}$ with predecessor i_j. We call σ the* common predecessor function *of I with respect to $(i_j)_{j \in J_s^{or}}$.*

2. *If $\sigma : J_s^{or} \to 2^{J_s}$ is a common predecessor function of I with respect to $(i_j)_{j\in J_s^{or}}$, we call $(i_j)_{j\in J_s^{or}}$* generating representatives *of σ.*
3. *Let $I' = (J, (\psi'_j)_{j\in J})$ be the $P|prec,\ p_j = 1|\gamma$ instance with $\psi'_j = \psi_j$ if $j \notin J_s^{or}$ and $\psi_j = x_{i_j}$ if $j \in J_s^{or}$. We call I' the and-*variation *of I with respect to $(i_j)_{j\in J_s^{or}}$.*

Observe the following properties of a common predecessor function $\sigma : J_s^{or} \to 2^{J_s}$: First, $j \in \sigma(j)$ for all $j \in J_s^{or}$. If $j, k \in J_s^{or}$ and $k \in \sigma(j)$, then $\sigma(j) = \sigma(k)$. If σ is the common predecessor function of I with respect to $(i_j)_{j\in J_s^{or}}$, then i_j is a predecessor of every $k \in \sigma(j)$, $j \in J_s^{or}$. Lastly, note that σ may have different generating representatives $(i_j)_{j\in J_s^{or}}$, $(i'_j)_{j\in J_s^{or}}$, $(i_j)_{j\in J_s^{or}} \neq (i'_j)_{j\in J_s^{or}}$. The following lemma proves that for a given common predecessor function, the choice of generating representatives is irrelevant (under certain conditions).

▶ **Lemma 7.** *Let $I = (J, (\psi_j)_{j\in J}, m)$ be a $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ instance, $\gamma \in \{C_{\max}, \sum_j C_j\}$, with successor set $J_s = J_s^{or} \dot\cup J_s^{and}$. Let $\sigma, \sigma' : J_s^{or} \to 2^{J_s}$ be common predecessor functions of I with respect to $(i_j)_{j\in J_s^{or}}, (i'_j)_{j\in J_s^{or}}$ respectively. Let L, L' be the and-variations of I with respect to $(i_j)_{j\in J_s^{or}}, (i'_j)_{j\in J_s^{or}}$ respectively. If*

$$I_s := \{j \in J_s^{or} \mid i_j \in J_s\} = \{j \in J_s^{or} \mid i'_j \in J_s\} \text{ and } i_j = i'_j \text{ for all } j \in I_s,$$

then L is feasible precisely if L' is feasible. In this case, $OPT(L') = OPT(L)$ if moreover $\sigma = \sigma'$ and $i_j = i'_k$ whenever $i'_j = i_k$, $j, k \in J_s^{or}$.

Proof. Assume that L is infeasible and let $C = (j_1, \ldots, j_r)$ be a cycle in the precedence graph, $j_0 := j_r$. Consider a job j_ℓ in the cycle, $\ell \in \{1, \ldots, r\}$. If $j_\ell \in J_s^{and}$ is an *and*-job in I, then $j_{\ell-1}$ is also a predecessor of j_ℓ in L'. If $j_\ell \in J_s^{or}$, then $j_{\ell-1} = i_j$. Since $j_{\ell-1}$ is a successor in L, it is also a successor in I and thus $j_{\ell-1} = i_j = i'_j$ is also a predecessor of j_ℓ in L'. Therefore, the precedence graph of L' contains the same cycle C and L' is infeasible.

Assume now that L is feasible, $\sigma = \sigma'$ and $i_j = i'_k$ whenever $i'_j = i_k$, $j, k \in J_s^{or}$. Let $(C_j)_{j\in J}$ be the completion times of an optimal schedule $\mathcal{S}$. We define a schedule $\mathcal{S}'$ with completion times $(C'_j)_{j\in J}$ for L'. We denote the sets in $\{\sigma(j) \cap J_s^{or} \mid j \in J_s^{or}\}$ by $S_1, \ldots, S_r$. Observe that $(S_\ell)_{\ell=1}^r$ is a partition of J_s^{or} and that $S_\ell = \sigma(j) \cap J_s^{or}$ precisely if $S_\ell = \sigma'(j) \cap J_s^{or}$, $j \in J_s^{or}, \ell = 1, \ldots, r$. For each $\ell = 1, \ldots, r$, let $i(\ell),\ i'(\ell) \in J_s^{or}$ be the common predecessor with $i(\ell) = i_k$, $i'(\ell) = i'_k$ for all $k \in S_\ell$. We obtain $\mathcal{S}'$ from $\mathcal{S}$ by swapping the positions of $i(\ell)$ and $i'(\ell)$ for all $\ell = 1, \ldots, r$. Observe that this is in fact the same as setting $C'_{i'_k} = C_{i_k}$ and $C'_{i_k} = C_{i'_k}$ for $k \in J_s^{or}$. All other jobs $j \in J$ remain as in $\mathcal{S}$, i.e. $C'_j = C_j$. Note that the schedule $\mathcal{S}'$ is well-defined, since we require $i'(\ell) = i(p)$ if $i(\ell) = i'(p)$ for $\ell, p = 1, \ldots, r$. Clearly, $\gamma(\mathcal{S}) = \gamma(\mathcal{S}')$. It remains to show that $\mathcal{S}'$ is feasible. To this end, let $k \in J_s$ be a successor job in I. Observe that $C_k = C'_k$: If $k \neq i_\ell, i'_\ell$ for all $\ell \in J_s^{or}$, then $C'_k = C_k$ by definition of C_k. Otherwise, if $k = i_\ell$ or $k = i'_\ell$ for some $\ell \in J_s^{or}$, then $\ell \in I_s$ and thus $k = i_\ell = i'_\ell$ and $C'_k = C_k$. Assume now that $k \in J_s^{and}$ is an *and*-job in I and let $j \in J$ be a predecessor of k in L'. Clearly, j is also a predecessor of k in L and thus $C_j < C_k$. If $C'_j = C_j$, then j precedes k in $\mathcal{S}'$ because $C'_j = C_j < C_k = C'_k$. Otherwise, $j = i_\ell$ or $j = i'_\ell$ for some $\ell \in J_s^{or}$ with $i_\ell \neq i'_\ell$. Assume first that $j = i_\ell$. Since k is an *and*-job in I with predecessor $i_\ell = j$, it follows that $k \in \sigma(\ell) = \sigma'(\ell)$. By definition of σ', job i'_ℓ is a predecessor of k in I and L and thus $C'_j = C_{i'_\ell} < C_k = C'_k$. We can show analogously that $C'_j < C'_k$ if $j = i'_\ell$. Finally, let $k \in J_s^{or}$ be a former *or*-job. By definition of L', the job i'_k is the only predecessor of k in L' and i_k is the only predecessor of k in L. Subsequently, $C'_{i'_k} = C_{i_k} < C_k = C'_k$, which completes the proof. ◀

We now describe the algorithm AND+OR-ALG-SUCC for $P|and\text{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$, in more detail. First, AND+OR-ALG-SUCC picks completion times C_j, $j \in J_s$, for the successors in the same way as AND-ALG-SUCC. For every *or*-job $j \in J_s^{or}$ with a predecessor $i \in J_s$ with $C_i < C_j$, we set $i_j = i$. Afterwards, AND+OR-ALG-SUCC iterates over all *or*-jobs $j_1, \dots, j_q \in J_s^{or}$ in order of non-decreasing completion time $C_{j_1} \leq \dots \leq C_{j_q}$. If no i_{j_i} has been chosen yet, we pick a subset $S_j \subseteq \{j_i, \dots, j_q\} \cup J_s^{and}$ of successor jobs with $j \in S_j$. Then we pick a job $i \in J$ such that the successors of i in $\{j_i, \dots, j_q\} \cup J_s^{and}$ are precisely the vertices in S_j. We set $i_k = i$ for all $k \in S_j \cap J_s^{or}$. (If such a job i does not exist, S_j is an infeasible choice.) Finally, the algorithm AND+OR-ALG-SUCC proceeds along the same lines as AND-ALG-SUCC for the *and*-variation of I w.r.t. $(i_j)_{j \in J_s^{or}}$.

▶ **Theorem 8.** *Let I be a feasible instance of $P|and\text{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ with k_s successors. Then* AND+OR-ALG-SUCC *returns in $\mathcal{O}(k_s^{k_s} \cdot 2^{k_s^2} \cdot m \cdot n^3)$ time the completion times of an optimal schedule $\mathcal{S}^*$.*

Proof. Let $J_s = J_s^{or} \dot\cup J_s^{and}$ be the set of successors with $|J_s| = k_s$ and $q := |J_s^{or}|$. By Lemma 3, there exists an optimal schedule $\mathcal{S}^* = (C_j^*)_{j \in J}$ such that $(C_{i_1}^*, \dots, C_{i_{k_p}}^*) \in \mathcal{C}$. Consider the iteration of AND+OR-ALG-SUCC where $C_j = C_j^*$ for all $j \in J_s$ and let $\mathcal{S}$ denote the schedule defined by $(C_j)_j$. Let $j_1, \dots, j_q$ be the ordering, in which AND+OR-ALG-SUCC iterates over the *or*-jobs. We denote by I_s the set of *or*-jobs $j \in J_s^{or}$, to which AND+OR-ALG-SUCC assigns a predecessor $i_j \in J_s$.

The idea of the proof is the following: First, we pick $(i_j^*)_{j \in J_s^{or}}$ such that i_j^* is a predecessor of j with $C_{i_j^*}^* < C_j^*$, $j \in J_s^{or}$. We argue, that for a certain choice of $(S_j)_j$, the algorithm computes representatives $(i_j)_{j \in J_s^{or}}$ such that $(i_j)_j, (i_j^*)_j$ as well as the associated common predecessor functions σ, σ^* comply with the conditions in Lemma 7.

More precisely, we pick $(i_j^*)_{j \in J_s^{or}}$ as follows: Consider the *or*-jobs j in order $j = j_1, \dots, j_q$. We set $i_j^* = i_j$ for $j \in I_s$. If $j = j_i$ and no i_j^* has been chosen yet, we pick an arbitrary predecessor i_j^* with $C_{i_j^*}^* < C_j^*$ but set $i_k^* = i_j^*$ for all $k = j_{i+1}, \dots, j_q$, of which i_j^* is also a predecessor. We denote by $\sigma^* : J_s^{or} \to 2^{J_s}$ the common predecessor function of I w.r.t. $(i_j^*)_{j \in J_s^{or}}$ and consider the iteration where AND+OR-ALG-SUCC sets $S_{j_i} = \sigma^*(j_i) \backslash \{j_1, \dots, j_{i-1}\}$ for $i = 1, \dots, q$. Observe that whenever AND+OR-ALG-SUCC is required to pick i_j according to S_j, then $S_j = \sigma^*(j)$, otherwise i_j would have been chosen in an earlier iteration. Due to the choice of $(S_j)_j$ and the properties of $(i_j^*)_j$, AND+OR-ALG-SUCC can find an i_j for $j = j_r$ such that 1. i_j is a predecessor of all jobs in S_j and 2. i_j is not a predecessor of any job in $(J_s^{and} \cup \{j_{r+1}, \dots, j_q\}) \setminus S_j$. We denote by $\sigma : J_s^{or} \to 2^{J_s}$ the common predecessor function of I w.r.t. $(i_j)_{j \in J_s^{or}}$. Then it is easy to verify that $(i_j)_j, (i_j^*)_j, \sigma, \sigma^*$ comply with the conditions in Lemma 7, which completes the proof. ◀

We have shown that $P|or/and\text{-}prec,\ p_j = 1|\gamma$ is polynomial-time solvable if the number k_s of successors is bounded by a constant while $P|and\text{+}or\text{-}prec,\ p_j = 1|\gamma$ is even FPT with respect to the number of successors. Recall that Berit [11] proved the *or*-constrained problem $P|or,\ p_j = 1|\gamma$ to be polynomial-time solvable, even if no restrictions are made on the number of successors. Naturally, three questions arise:

1. Is $P|or/and\text{-}prec,\ p_j = 1|\gamma$ FPT when parameterized by k_s? We give a negative answer to this question by proving that $P|or/and\text{-}prec,\ p_j = 1|\gamma$ is $W[1]$-hard (see Theorem 9).
2. Is $P|and/or\text{-}prec,\ p_j = 1|\gamma$ polynomial-time solvable if the number k_s of successors is bounded by a constant? We will give a negative answer to this question by proving that $P|and/or\text{-}prec,\ p_j = 1|\gamma$ is $\mathcal{NP}$-hard, even if $k_s = 1$ (see Proposition 11).

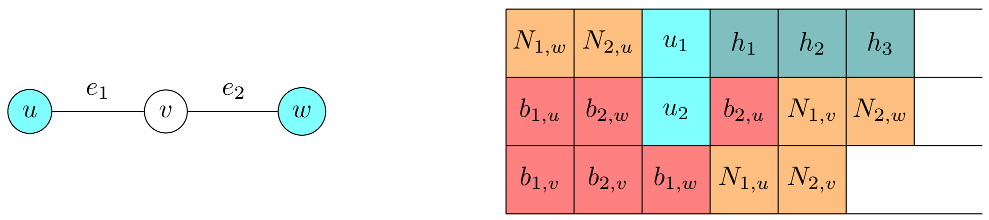


Figure 2 Reduction from Independent Set with $k = 2$ to $P|or/and\text{-}prec, p_j = 1|\gamma$ with $k_s = 2k+1$. Blue jobs and preceding orange jobs represent the size and selection of an independent set. In schedules with $C_{max} = 2k + 1$, green (red) jobs are used to block space after (before) blue jobs.

3. Is $P|and\text{+}or\text{-}prec,\ p_j = 1|\gamma$ FPT with respect to the number of *and*-successors? Surprisingly, $P2|and\text{+}or\text{-}prec,\ p_j = 1|\gamma$ turns out to be para-$\mathcal{NP}$-hard when parameterized by the number of *and*-successors. In fact, we prove that the problem is $\mathcal{NP}$-hard, even if there is only a single *and*-successor (see Theorem 13).

▶ **Theorem 9.** *$P|or/and\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is $W[1]$-hard when parameterized by the number k_s of successors.*

Proof. We prove the claim by an FPT-reduction from Independent Set. Let $I' = (G, k)$ be an instance of Independent Set where $G = (V, E)$ is an undirected graph with $n' := |V|$ nodes and $k \in \mathbb{N}_0$ is the required size of an independent set, w.l.o.g. $k < n'$. By

$$\overline{\mathcal{N}}(v) := \{w \in V \mid \{v, w\} \in E\} \cup \{v\}$$

we denote the closed neighborhood of a vertex $v \in V$. We construct from I' an instance I of $P|or/and\text{-}prec,\ p_j = 1|C_{\max}$ with n' machines and require a makespan of at most $2(k + 1)$. An interpretation of the construction is given below. We introduce $2(k + k \cdot n') + 1$ jobs by setting

$$J := \{u_i \mid i = 1, \ldots, k\} \cup \{N_{iv}, b_{iv} \mid i = 1, \ldots, k, v \in V\} \cup \{h_i \mid i = 1, \ldots, k + 1\}.$$

The precedence constraints are defined as follows: Jobs N_{iv}, b_{iv} in the second subset do not have predecessors. The auxiliary jobs $h_1, \ldots, h_{k+1}$ have to be scheduled in a chain that follows the completion of all u_i, $i = 1, \ldots, k$. That is, $\psi_{h_{i+1}} = x_{h_i}$ for all $i = 1, \ldots, k$ and $\psi_{h_1} = \bigwedge_{i=1}^{k} x_{u_i}$. The precedence constraint of a job u_i is the disjunction $\psi_{u_i} = \bigvee_{v \in V} K_{iv}$ of n' clauses K_{iv}, where

$$K_{iv} := \bigwedge_{u \notin \overline{\mathcal{N}}(v)} x_{N_{iu}} \wedge \bigwedge_{u \in \overline{\mathcal{N}}(v)} x_{b_{iu}} \wedge \bigwedge_{j<i} x_{N_{jv}},\ i = 1, \ldots, k, v \in V.$$

We interpret $v \in V$ as the i-th node in the independent set if K_{iv} is satisfied by the jobs that precede u_i, $i = 1, \ldots, k$. In a schedule with makespan $2(k + 1)$, the chain of the auxiliary jobs $h_1, \ldots, h_{k+1}$ guarantees that all jobs $u_1, \ldots, u_k$ have to be completed by time $k + 1$. This leaves a total of at most $k \cdot n'$ time slots to schedule the predecessors that are needed to make $u_1, \ldots, u_k$ available. However, each clause K_{iv} is composed of $n' + i - 1$ literals in a way that each of the last $i - 1$ predecessors N_{jv}, $j = 1, \ldots, i - 1$, has to be a common predecessor with u_j, due to the space restrictions. Due to this, a node v_i for which K_{iv_i} is satisfied is disjoint and non-adjacent to the nodes v_j for which the previous $i - 1$ clauses K_{jv_j} are satisfied, $j = 1, \ldots, i - 1$. Figure 2 depicts an example of the construction. Since an optimal schedule for $\gamma = \sum_j C_j$ simultaneously optimizes the makespan, the hardness for the sum of completion times follows too. ◀

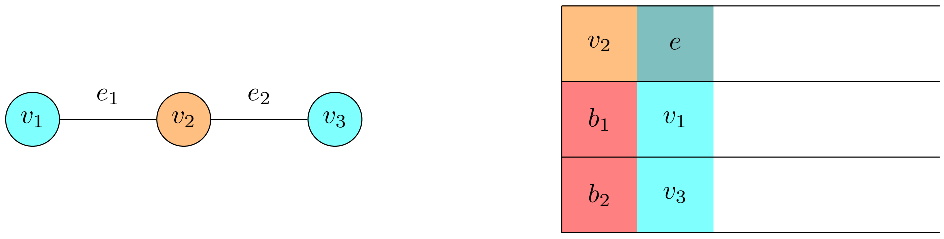


Figure 3 Reduction of Vertex Cover with $n' = 3$ and $k = 1$ to $P|and/or\text{-}prec, p_j = 1|\gamma$ with $k_s = 1$.

From the proof of Theorem 9, we moreover get the following corollary:

▶ **Corollary 10.** *$P|or/and\text{-}prec,\ p_j = 1|C_{\max}$ is $W[1]$-hard if parameterized by $C_{\max}$ and k_s.*

Given the complexity result in Theorem 9, the existence of a fixed-parameter tractable algorithm for $P|or/and\text{-}prec,\ p_j = 1|\gamma$ parameterized by k_s is unlikely. In the context of *and/or*-constraints, we provide even stronger evidence against the fixed-parameter tractability of $P|and/or\text{-}prec,\ p_j = 1|\gamma$ parameterized by k_s, by proving that the problem is para-$\mathcal{NP}$-hard.

▶ **Proposition 11.** *$P|and/or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$ is $\mathcal{NP}$-hard, even if $k_s = 1$.*

Proof. We prove the claim by a reduction from Vertex Cover. Let $I' = (G, k)$ be an instance of Vertex Cover, where $G = (V, E)$ is an undirected Graph and $k \in \mathbb{N}$ is the required size of a vertex cover. We may assume that $k < |V| =: n'$, otherwise I is trivial. We construct from I' an instance I of $P|and/or\text{-}prec,\ p_j = 1|\gamma$ with a single successor. We consider $m := n'$ machines. The set J of jobs is composed of a job v for each vertex $v \in V$, a single job e, which will be the only successor and $n' - k$ auxiliary jobs $b_1, ..., b_{n'-k}$. Through precedence constraints, we force all auxiliary jobs as well as one end-vertex per edge to be processed before e: We set

$$\psi_e = \bigwedge_{i=1}^{n'-k} x_{b_i} \wedge \bigwedge_{\{v,w\} \in E} (x_v \vee x_w)$$

and $\psi_j = \mathbb{1}$ for all $j \in J \setminus \{e\}$. We require to find a feasible schedule $\mathcal{S}$ with $C_{\max} \leq 2$ if $\gamma = C_{\max}$ or $\sum_{j \in J} C_j \leq 3n' - 2k + 2$ if $\gamma = \sum_j C_j$. It is easy to see that I' is a yes-instance precisely if we can find a schedule for I', which processes all auxiliary jobs and a vertex cover of size k during the first time slot and all remaining jobs during time slot two. The construction is visualized in Figure 3. ◀

From the proof of Proposition 11, we also get the following corollary:

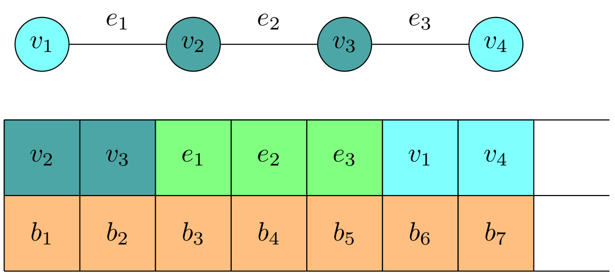
▶ **Corollary 12.** *$P|and/or\text{-}prec,\ p_j = 1|C_{\max}$ is para-$\mathcal{NP}$-hard, if parameterized by $C_{\max}$ and k_s.*

5 Parameterization by the number of machines

A classical parameter in the study of parameterized complexity of scheduling problems, is the number of machines. In [4], Coffman and Graham prove that the $\mathcal{NP}$-hard problem $P|prec,\ p_j = 1|C_{\max}$ becomes polynomial when restricted to two machines. For an arbitrary but constant number of machines, the complexity is open. However, Bodlaender and

Fellows [2] prove that $P|prec,\ p_j = 1|C_{\max}$ is $W[2]$-hard when parameterized by the number of machines. In the following, we prove that $P2|and{+}or\text{-}prec,\ p_j = 1|\gamma$ is $\mathcal{NP}$-hard for $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ and thus $P|and{+}or\text{-}prec,\ p_j = 1|\gamma$ is para-$\mathcal{NP}$-hard when parameterized by the number of machines. This result is also interesting in the context of the interplay between *and*- and *or*-constraints: On the one hand, it shows that the introduction of *or*-constraints turns the polynomial-time solvable $P2|prec,\ p_j = 1|C_{\max}$ into an $\mathcal{NP}$-hard problem. On the other hand, the result below shows that the introduction of a single *and*-job turns the polynomial-time solvable $P|or\text{-}prec,\ p_j = 1|C_{\max}$ into an $\mathcal{NP}$-hard problem. For an *and+or*-constrained instance with successors in $J_s = J_s^{or} \dot\cup J_s^{and}$, we denote by $k_s^{and} := |J_s^{and}|$ the number of *and*-successors.

▶ **Theorem 13.** *$P2|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j, \sum_j w_j C_j\}$ with $k_s^{and} = 1$ is $\mathcal{NP}$-hard.*



e_1
e_2
e_3
v_1
v_2
v_3
v_4
v_2 v_3 e_1 e_2 e_3 v_1 v_4
b_1 b_2 b_3 b_4 b_5 b_6 b_7

Figure 4 Reduction from Vertex Cover to $P2|and{+}or\text{-}prec, p_j = 1|\gamma$ with $J_s^{and} = \{b_6\}$.

Proof. We prove the claim by a reduction from Vertex Cover. Let $I = (G, k)$ be an instance of Vertex Cover, where $G = (V, E)$ is an undirected graph and k is the requested size of a vertex cover. We assume that $k < |V|$, otherwise I is trivial. We construct from I an instance I' of $P2|and{+}or\text{-}prec,\ p_j = 1|\gamma$, $\gamma \in \{C_{\max}, \sum_j C_j\}$, where the idea is the following: We introduce one job per edge, one job per vertex and $\ell := |V| + |E|$ auxiliary jobs to block the second machine. Using precedence constraints, we will force a feasible schedule to process all jobs in a vertex cover of size at most k before finishing the edge jobs.

More precisely, we set $J := V \cup E \cup B$ where $B = \{b_1, \dots, b_\ell\}$ is the set of auxiliary jobs. All vertex jobs, as well as the first auxiliary job, may start at any time, i.e. $\psi_j = \mathbb{1}$ if $j \in V \cup \{b_1\}$. Edges may not be scheduled unless one of their end-vertices has been scheduled already, i.e. $\psi_e = x_u \vee x_v$ for $e = \{u, v\} \in E$. We set

$$\psi_{b_{k+|E|+1}} = x_{b_{k+|E|}} \wedge \bigwedge_{e \in E} x_e$$

to make sure that all edge jobs have been scheduled by the time we start the $(k + |E| + 1)$-st auxiliary job. To process all auxiliary jobs in a chain, we set $\psi_{b_i} = x_{b_{i-1}}$, $i \in \{2, \dots, \ell\} \setminus \{k + |E| + 1\}$. Observe that $J_s^{and} := \{b_{k+|E|+1}\}$ and $J_s^{or} := E \cup \{b_2, \dots, b_{k+|E|}, b_{k+|E|+2}, \dots, b_\ell\}$ is a partition of the successors into *or*- and *and*-jobs. The construction is visualized in Figure 4. It is easy to see that I is a yes-instance, precisely if there exists a schedule for I' with $C_{\max} = \ell$, $(\sum_j C_j = \ell \cdot (\ell + 1))$. ◀

6 Summary and Outlook

In this paper, we study the problems of minimizing the makespan and the sum of completion times in parallel machine scheduling of unit-time jobs under precedence constraints. In our setting, each job j is associated with a precedence constraint ψ_j, encoded as a Boolean formula over the set of jobs. A schedule satisfies ψ_j if setting exactly the jobs scheduled before j to true satisfies the formula. Depending on the structure of these formulas, we distinguish between *and*-, *or*-, *and+or*-, *or/and*-, and *and/or*-constraints.

To analyze the parameterized complexity of these problems, we introduce two new structural parameters: the total number of predecessors and the total number of successors. Restricting the number of predecessors yields fixed-parameter tractability, even for general precedence constraints. In contrast, restricting the number of successors leads to a more nuanced landscape: for traditional *and*-constraints and more general *and+or*-constraints, the problem is FPT, but for *or/and*-constraints (DNF), it becomes $W[1]$-hard. For and/or-constraints (CNF), the problem is even para-$\mathcal{NP}$-hard. A common parameter in the analysis of scheduling problems is the number of machines. While minimizing the makespan for *and*-constrained unit jobs is known to be $W[2]$-hard parameterized by the number of machines (Bodlaender and Fellows [2]), we show that the same problem is para-$\mathcal{NP}$-hard in the presence of *and+or*-constraints.

We believe these results open up several promising research directions. One such direction concerns the interplay between *and*- and *or*-constraints. Naturally, any known hardness result for problems with classical *and*-constraints carries over to the more general version of the problem with *and+or*-constraints. Conversely, certain *or*-constrained problems can be solved in polynomial time, while they are $\mathcal{NP}$-hard in the *and*-constrained version (Berit [11]). However, Theorem 13 shows that it is inaccurate to think that the hardness of *and+or*-constrained problems stems only from the *and*-constraints: While minimizing the makespan or sum of completion times for unit jobs on two machines is in $\mathcal{P}$, both in the *or*- and the *and*-constrained version (Coffman and Graham [4], Berit [11]), we show that scheduling with *and+or*-constraints on just two machines is $\mathcal{NP}$-hard. We suggest studying parameters that measure the ratio between *or*- and *and*-constraints. In Theorem 13, we show that $P2|and\text{+}or\text{-}prec,\ p_j = 1|C_{\max}$ is para-$\mathcal{NP}$-hard parameterized by the number of *and*-successors, while Theorem 8 proves the same problem to be FPT parameterized by the total number of successors. Open questions in the same spirit include whether restricting only the predecessors of *and*-jobs or only the number of *or*-successors on two machines leads to tractability or retains hardness.

Table 1 Overview of parameterized complexity results for parallel machine scheduling of unit-time jobs with generalized precedence constraints.

Problem	γ	Parameter	Complexity	Source
$P\ \|gen\text{-}prec,\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_p	in $\mathcal{FPT}$	Thm 2
$P\ \|gen\text{-}prec,\ p_j = 1\|\gamma$	$\sum_j w_j C_j$	k_p	in $\mathcal{XP}$	Thm 2
$P\ \|and + or\text{-}prec,\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_s	in $\mathcal{FPT}$	Thm 8
$P2\|and + or\text{-}prec,\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j,\ \sum_j C_j$	–	$\mathcal{NP}$-hard	Thm 13
$P\ \|and + or\text{-}prec,\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j,\ \sum_j C_j$	$m,\ k_s^{and}$	para-$\mathcal{NP}$-hard	Thm 13
$P\ \|or/and\text{-}prec,\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_s	in $\mathcal{XP}$	Cor 5
$P\ \|or/and\text{-}prec,\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j,\ \sum_j w_j C_j$	k_s	$W[1]$-hard	Thm 9
$P\ \|and/or\text{-}prec,\ p_j = 1\|\gamma$	$C_{\max},\ \sum_j C_j$	k_s	para-$\mathcal{NP}$-hard	Prop 11

Moreover, our results show that it is fruitful to explore parameters that limit job dependencies, especially for problems that become hard due to precedence constraints. This perspective contrasts with the focus on the width in classical *and*-constrained problems, which instead promotes high interrelatedness among jobs. In *and*-constrained problems, parameters of the precedence graph, which reduce interrelatedness, may include the size of a largest clique or tournament and the largest out- or in-degree. In particular, we propose a hybrid approach: combining parameters that limit the portion of input influenced by precedence constraints with parameters that enforce high interrelatedness within affected jobs.

References

1 Stéphane Bessy and Rodolphe Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, 2019. doi:10.1007/S10951-018-0581-1.

2 Hans L Bodlaender and Michael R Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995. doi:10.1016/0167-6377(95)00031-9.

3 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, pages 22–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.22.

4 Edward G Coffman and Ronald L Graham. Optimal scheduling for two-processor systems. *Acta informatica*, 1:200–213, 1972. doi:10.1007/BF00288685.

5 Thomas Erlebach, Vanessa Kääb, and Rolf H Möhring. Scheduling and/or-networks on identical parallel machines. In *International workshop on approximation and online algorithms*, pages 123–136. Springer, 2003. doi:10.1007/978-3-540-24592-6_10.

6 Donald W Gillies and Jane W-S Liu. Scheduling tasks with and/or precedence constraints. *SIAM Journal on Computing*, 24(4):797–810, 1995. doi:10.1137/S0097539791218664.

7 Michael Goldwasser, J-C Latombe, and Rajeev Motwani. Complexity measures for assembly sequences. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1851–1857. IEEE, 1996.

8 Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

9 Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

10 Felix Happach. Makespan minimization with or-precedence constraints. *Journal of Scheduling*, 24(3):319–328, 2021. doi:10.1007/S10951-021-00687-6.

11 Berit Johannes. On the complexity of scheduling unit-time jobs with or-precedence constraints. *Operations research letters*, 33(6):587–596, 2005. doi:10.1016/J.ORL.2004.11.009.

12 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21:493–503, 2018. doi:10.1007/S10951-017-0550-0.

13 Sanghyup Lee, Ilkyeong Moon, Hyerim Bae, and Jion Kim. Flexible job-shop scheduling problems with "AND"/"OR" precedence constraints. *International Journal of Production Research*, 50(7):1979–2001, 2012.

14 Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978. doi:10.1287/OPRE.26.1.22.

15 Matthias Mnich and René Van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018. doi:10.1016/J.COR.2018.07.020.

16 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1):533–562, 2015. doi:10.1007/S10107-014-0830-9.

17 Rolf H Möhring, Martin Skutella, and Frederik Stork. Scheduling with and/or precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004. doi:10.1137/S009753970037727X.

18 Jesper Nederlof and Céline M. F. Swennenhuis. On the Fine-Grained Parameterized Complexity of Partial Scheduling to Minimize the Makespan. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2020.25.

19 Damien Prot and Odile Bellenguez-Morineau. A survey on how the structure of precedence constraints may change the complexity class of scheduling problems. *Journal of Scheduling*, 21(1):3–16, 2018. doi:10.1007/S10951-017-0519-Z.

20 Ravi Sethi. On the complexity of mean flow time scheduling. *Mathematics of Operations Research*, 2(4):320–330, 1977. doi:10.1287/MOOR.2.4.320.

21 René Van Bevern, Robert Bredereck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In *International conference on discrete optimization and operations research*, pages 105–120. Springer, 2016. doi:10.1007/978-3-319-44914-2_9.