Efficient Project Scheduling with Autonomous Learning Opportunities

Efficient Project Scheduling with Autonomous Learning Opportunities

Alessandro Hill
DEI, University of Bologna, alessandro.hill@unibo.it

Thomas W. M. Vossen
Leeds School of Business, University of Colorado Boulder, vossen@colorado.edu

We consider novel project scheduling problems in which the experience gained from completing selected activities can be used to accelerate subsequent activities. Given a set of potential learning opportunities, our model aims to identify the opportunities that result in a maximum reduction of the project makespan when scheduled in sequence. Accounting for the impact of such learning opportunities causes significant complications, due to the cyclic nature of the learning relations and their interference with the precedence network.

We propose additive and subtractive algorithms that iteratively reschedule the project using an enhanced topological sorting algorithm. Learning opportunities are integrated, activated and potentially deactivated in each step by maintaining the acyclicity of the combined precedence and learning network. To illustrate the challenges that arise in this setting, we first consider the special case where activities can learn from at most on other activity. Subsequently, we extend our approach to the general case that admits multiple learning opportunities. We show that our approaches guarantee the construction of an optimal solution in polynomial time.

In a computational study using 340 small and large resource-unconstrained PSPlib instances, we analyze the model behaviour under various scenarios of learning intensity and learning opportunity. We demonstrate that significant project speedups can be obtained when proactively accounting for learning opportunities.

Key words: project management; scheduling; autonomous learning

1. Introduction

The critical role of effectively scheduling activities when managing projects has long been established, and numerous theoretical contributions have supported the growth and adaptation of formal project management tools in practice. As organizations increasingly organize their processes through projects (Hall 2012), the importance of project scheduling is only bound to grow further. However, the increasing variety and novel applications of

project management also present new challenges and opportunities to improve project scheduling decisions that are not well understood.

In particular, we explore how to take advantage of learning opportunities when scheduling activities in a resource-unconstrained project. It is well-known that experience can lead to increased efficiency and that actively managing learning can lead to improved project performance. While the active management of learning within projects can extend to the *deliberate* learning that explicitly considers investments and resources allocated to learning (Cao et al. 2024), we concentrate on strategically accounting for the *autonomous* learning from experience (Biskup 2008) that may occur during project execution. The impact of learning effects has been studied for certain scheduling problems without resource constraints, such as single-machine scheduling problems (Lee et al. 2010, Qian and Steiner 2013, Bai et al. 2018); their impact on project scheduling decisions, however, has received little attention. To the best of our knowledge, this also holds for resource-constrained project scheduling (RCPSP), where learning effects can be incorporated as a special case of the multi-mode RCPSP (Peteghem and Vanhoucke 2015). However, the inherent complexity of the RCPSP makes it hard to isolate the impact and understand the complications of incorporating learning effects.

To capture learning opportunities, we express the well-known project scheduling problem (without resource constraints) using an Activity-On-Node network. We extend this with a so-called *learning network*, whose arcs present a novel optional precedence relationship that allows us to model learning effects. Any arc (i, j) in the learning network signifies that activity j can learn from activity i. Specifically, the duration of activity j can be reduced *provided that* it starts after activity i has been completed. We propose an efficient polynomial time algorithm for minimizing the makespan when such learning opportunities are present, and show that strategically accounting for learning opportunities can have a significant impact on project completion times.

Consequently, the main contributions of our paper can be summarized as follows:

- We propose a new approach to account for the impact of learning in project scheduling. Our approach relies on a novel type of precedence relationship to capture learning opportunities, which is distinct from other forms of generalized precedences that have been proposed in the literature.

- We establish an efficient algorithm to minimize the makespan when accounting for the impact of learning effects, and highlight its differences with organic learning approaches that do not explicitly account for learning opportunities during the scheduling phase.
- We capture and delineate the potential benefits of learning using an extensive computational study based on a comprehensive set of instances and a broad set of learning parameters.

The remainder of this paper is organized as follows. Section 2 surveys and reviews related work, while Section 3 introduces our model setup and notation. To illustrate the challenges that arise in our setting, Section 4 first considers the case where activities have at most one learning opportunity and describes the key ideas behind our approach for minimizing the makespan. In Section 5, we extend this approach to the more general setting where activities have multiple learning opportunities. Section 6 reports the computational analysis for both our optimization model and related approaches, which is followed by a summary of key insights and conclusions in Section 7.

2. Related Literature

Project scheduling has a long history, dating back to the development of the critical path method in the 1950s (Kelley Jr 1961). Project scheduling problems are defined by a given collection of activities and their durations, together with precedence requirements between those activities. When the objective is to minimize makespan, project schedules can be determined efficiently using a topological ordering of the activities. Similarly, the project scheduling problem can also be solved in polynomial time when the objective is to maximize the net present value or some other weighted function of the activity start times (Möhring et al. 1999). However, the research literature has predominantly focused on resource-constrained project scheduling problems (RCPSPs), where activities require the use of limited resources during their execution (Johnson 1967). RCPSPs are computationally challenging, and we refer to Hartmann and Briskorn (2010, 2022) and Schwindt et al. (2015) for a comprehensive survey of RCPSP variants and corresponding algorithmic approaches. Exact mathematical programming formulations have been studied since Pritsker et al. (1969), and time-indexed RCPSP formulations (Artigues 2017) have shown strong theoretical and computational performance. Over the last decade, however, constraint programming (CP) approaches have emerged as the dominant exact solution

methods for the RCPSP, and we refer to Laborie et al. (2018) for a review of CP methods for the RCPSP.

Ever since the seminal work by Ebbinghaus (1885) it is well-known that the acquisition of skills can be described by a learning curve. This concept spawned a vast body of research on the effects of learning for more than 140 years (Yelle 1979). In particular, production processes can become more efficient due to experience gains over time (Wright 1936). This also extends to scheduling problems, and the impact of learning effects has been studied extensively for machine scheduling problems (Bai et al. 2018, Lee et al. 2010, Qian and Steiner 2013). We refer to Biskup (2008) and Heuser et al. (2022) for an overview of scheduling problems that incorporate learning, and to Glock et al. (2019) for a more general review of applications that involve learning in production and operations management.

We believe that in many cases the addition of practical features such as setup times, time constraints or multiple modes - in contrast to our work - significantly increases the complexity of the resource-unconstrained PSP and, therefore, many relevant model extensions are studied in a resource-constrained setting. Research that considers the impact of learning for project scheduling problems almost exclusively focuses on the RCPSP. Van Peteghem and Vanhoucke (2015) incorporate learning effects through both reduced resource utilization and altered job durations using the discrete time/cost trade-off problem (DTCTP). The DTCTP is closely related to the RCPSP but restricted to a single non-renewable resource, and each job can be executed in one of multiple predefined modes. These modes are used to represent the execution of a job consuming increased amounts of resources at augmented speed, following an underlying (resource) learning curve. Korytkowski and Malachowski (2019) consider the estimation of activity durations when learning impacts project performance, and use these estimates in a multi-skill variant of the RCPSP. Using an IT project as an example, they show that accounting for learning effects improves estimation of project durations. Hill et al. (2021) use CP to solve hard RCPSP instances of different sizes where learning impacts activity durations. While our work uses the same approach as theirs to account for learning effects, we prove that the makespan can be minimized efficiently in the absence of resource constraints. This in turn can be useful to obtain strong lower bounds for constraint programming approaches. A recent paper by Cao et al. (2024) also considers RCPSPs with *deliberate* learning. In contrast to the autonomous learning we

consider, their model explicitly accounts for investments and resources allocated to learning. For this setting, the authors propose heuristic solution methods that are evaluated using a case study.

Our approach also relates to other streams of research in project scheduling. First, we introduce a novel conditional precedence relationship to capture learning opportunities. As such, our research extends previous work that considers generalized precedence relationships for project scheduling problems. Generalized precedence relationships were introduced in Elmaghraby and Kamburowski (1992), and further considered by Reyck et al. (1999) and De Reyck and Herroelen (1999) for RCPSPs. Möhring et al. (2004) consider project scheduling problems with and/or precedence relations, which also allows the possibility that activities can start when at least one of its predecessors has finished. However, the combination of conditional relationships together with a reduction in activity durations in our setting is different from previous approaches.

In addition, our work is related to the literature on project crashing (Babu and Suresh 1996, Hochbaum 2016) which considers the tradeoff between completion time and costs when scheduling projects. Similarly, project fast tracking is another technique to reduce project completion times by allowing certain (subparts of) activities to be completed in parallel when they would ordinarily be done sequentially (Vanhoucke 2008, Vanhoucke and Debels 2008). In our model, however, project completion times are reduced through sequencing that accounts for lower activity durations for the activities that learn without considering costs. In that regard, our approach is also different from project scheduling models that consider sequence-dependent setup costs (Dodin and Elimam 1997).

Finally, we note that our model can also be interpreted as a certain multi-mode project scheduling model. Multi-mode project scheduling models have been used in resource-constrained settings to allow an activity to be executed in one of several modes which differ in duration and resource-consumption. This concept has typically been considered to model the trade-off between duration and resource consumption (Hartmann and Briskorn 2022). While the impact of learning on activity durations in our model could be represented by using separate modes, our model setup would also require mode-dependent precedences. As existing approaches for multi-mode project scheduling commonly define identical precedences for all modes of a job, they are generally ill-equipped to handle the sequence-dependent activity durations that arise when incorporating learning effects.

3. Project Scheduling with Learning Opportunities

We consider a directed acyclic graph $\mathcal{G}^{\mathcal{A}} = (\mathcal{V}, \mathcal{A})$. Nodes in the set $\mathcal{V}$ represent activities that need to be completed, while the arcs $(i,j) \in \mathcal{A}$ represent precedences. Each activity $j \in \mathcal{V}$ has a duration d_j. We assume that $\mathcal{G}^{\mathcal{A}}$ includes a unique source node s (d_s=0) and a unique sink node t (d_t=0).

In addition, let $\mathcal{L}$ be a given collection of learning opportunities that can be represented by the directed graph $\mathcal{G}^{\mathcal{L}} = (\mathcal{V}, \mathcal{L})$. Each learning opportunity $(i,j) \in \mathcal{L}$ implies that activity j can learn from activity i, that is, the duration of activity j is reduced if activity i is completed before or at the start time of activity j. For each learning opportunity $(i,j) \in \mathcal{L}$, we refer to i as the teacher and j as the learner.

In our most general case, each learner activity can have *multiple* teachers and we define $\mathcal{L}_j$ to be the set of arcs in $\mathcal{G}^{\mathcal{L}}$ that end at activity j. To express the reduction in activity durations that occur due to learning, we use the functions

$$l_j(L) = l_j(L \cap \mathcal{L}_j), \qquad \forall j \in \mathcal{V}, L \subseteq \mathcal{L}.$$

We assume that $l_j(\varnothing) = 0$ and that $l_j(\mathcal{L}) < d_j$. In addition, we also assume that $l_j(L)$ is monotone non-decreasing in L. This allows us to capture a wide range of learning curve models, which are typically characterized by the fact that experience improves performance (Glock et al. 2019).

We will first consider the special case where $|\mathcal{L}_j| \leqslant 1$ for each $j \in \mathcal{V}$, that is, there is at most one learning opportunity for each activity j. Hence, the indegree of each node in $\mathcal{G}^{\mathcal{L}}$ is at most one. When considering this situation, we omit the argument in the learning function and simply use l_j ($0 < l_j < d_j$) to express the reduction in the duration of activity j when activity i is completed before or at the start time of activity j.

Finally, we use $\mathcal{G}^{\mathcal{A}+\mathcal{L}} = (\mathcal{V}, \mathcal{A} \cup \mathcal{L})$ to represent the combined network containing both precedences and learning opportunities. We also use $\mathcal{G}^{\mathcal{A}}_{CL} = (\mathcal{V}, \mathcal{A}_{CL})$ to represent the transitive closure of $\mathcal{G}^{\mathcal{A}}$, and further assume that $(i,j) \in \mathcal{L}$ implies that $(i,j) \notin \mathcal{A}_{CL}$ (no free learning) and that $(j,i) \notin \mathcal{A}_{CL}$ (no impossible learning). In Table 1, we summarize the notation used throughout this paper.

Table 1 The symbols and abbreviations used in this paper.

Symbol	Explanation
$\mathcal{V}$	Set of activities.
s	Source (start) activity.
t	Sink (terminal) activity.
$\mathcal{A}$	Set of precedences.
$\mathcal{A}_{CL}$	Transitive closure of $\mathcal{A}$.
$\mathcal{L}$	Set of learning opportunities.
$\mathcal{L}_j$	Learning opportunities in $\mathcal{L}$ for activity $j \in \mathcal{V}$ (PSP-ML).
$\mathcal{G}^{\mathcal{A}}$	Digraph induced by $\mathcal{A}$.
$\mathcal{G}^{\mathcal{L}}$	Digraph induced by $\mathcal{L}$.
$\mathcal{G}^{\mathcal{A}+\mathcal{L}}$	Digraph induced by $\mathcal{A}$ and $\mathcal{L}$.
d_j	Duration of activity $j \in \mathcal{V}$.
$l_j(L)$	Learning benefit for activity $j \in \mathcal{V}$ and learning opportunities $L \subseteq \mathcal{L}_j$ (PSP-ML).
l_j	Learning benefit for activity $j \in \mathcal{V}$ (PSP-SL).

4. Project Scheduling with a Single Learning Opportunity

To illustrate the challenges that arise in project scheduling problems that involve learning and highlight the key ideas behind our approach, we first consider project scheduling problems when there is at most one learning opportunity for each activity. To establish a formal problem statement, we start by considering the case where $\mathcal{L} = \varnothing$. This corresponds to the classical project scheduling problem (PSP), where the minimum makespan can be determined by finding a longest path in $\mathcal{G}^{\mathcal{A}}$. Throughout this work, we will use variables $u_j \geqslant 0$ to represent the completion time of activity j for all $j \in \mathcal{V}$. A natural (dual) linear programming (LP) formulation for the PSP is as follows:

$$z^D = \min \; u_t - u_s$$

$$\text{s.t. } u_j - u_i \geqslant d_j, \qquad \forall (i,j) \in \mathcal{A}.$$

Here, we can set $u_s = 0$ without loss of optimality. Given that $\mathcal{G}^{\mathcal{A}}$ is acyclic, a simple way to solve the problem is to consider a topological ordering (numbering) of its nodes. Specifically, let TopoSort be a bijective function that topologically orders the nodes of $\mathcal{G}^{\mathcal{A}}$, that is, $\text{TopoSort}(\mathcal{V}, \mathcal{A}) : \{1, \ldots, |\mathcal{V}|\} \to \mathcal{V}$. For a given topological ordering of the precedence graph $\text{TopoSort}(\mathcal{V}, \mathcal{A})$, $\tau(k)$ will represent the node that is ordered at k^{th} position. Furthermore,

this ordering has the property that there exists no arc $(\tau(k'),\tau(k))$ in $\mathcal{G}^{\mathcal{A}}$ if $k' > k$. To derive an optimal schedule, we can apply the following Bellman equations for a given ordering τ:

$$u_{\tau(1)} = 0,$$

$$u_{\tau(k)} = \max_{i:(i,\tau(k))\in\mathcal{A}} u_i + d_{\tau(k)}, \qquad \forall k \in 2,\ldots,|\mathcal{V}|.$$

We observe that the resulting schedule is also optimal regarding the completion time of each individual activity $j \in \mathcal{V}$, in that there exists no schedule where j has a completion time that is strictly less than u_j.

In the project scheduling problem with a single learning opportunity (PSP-SL), we extend the PSP by considering the impact of learning. Similar to the PSP, the aim is to find a longest path. In addition, however, we aim to identify an optimal subset $L \subseteq \mathcal{L}$ of learning opportunities. For a given set of learning opportunities L, we define the set of all learner activities as $\mathcal{V}(L) = \{j \in \mathcal{V} : \exists_{i\in\mathcal{V}}(i,j) \in L\}$. Then the following (dual) formulation can be used to determine a longest path:

$$z^D(L) = \min\ u_t - u_s$$

$$\text{s.t. } u_j - u_i \geqslant \begin{cases} d_j, & j \notin \mathcal{V}(L); \\ d_j - l_j, & j \in \mathcal{V}(L), \end{cases} \qquad \forall (i,j) \in \mathcal{A} \cup L.$$

Its corresponding primal formulation equals

$$z^P(L) = \max \sum_{(i,j)\in\mathcal{A}\cup L: j\notin\mathcal{V}(L)} d_j x_{ij} + \sum_{(i,j)\in\mathcal{A}\cup L: j\in\mathcal{V}(L)} (d_j - l_j) x_{ij}$$

$$\text{s.t.} \sum_{i:(i,j)\in\mathcal{A}\cup L} x_{ij} - \sum_{i:(j,i)\in\mathcal{A}\cup L} x_{ij} = \begin{cases} -1, & j = s; \\ 1, & j = t; \\ 0, & \text{otherwise}, \end{cases} \qquad \forall j \in \mathcal{V},$$

$$x_{ij} \geqslant 0 \qquad \forall (i,j) \in \mathcal{A} \cup L.$$

Each continuous variable x_{ij} is associated with the constraint for arc (i,j) in the dual formulation above. Observe that the dual formulation will be infeasible if $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ contains a cycle given the assumption that $d_j - l_j > 0$ for all $j \in \mathcal{V}\backslash\{s,t\}$. In that case, the primal is unbounded and we have $z^P(L) = \infty$. The overall problem is therefore to determine a set

of learning opportunities such that the longest path is minimized. We can formulate the resulting problem as

$$\textbf{PSP-SL}: \quad z = \min_{L \subseteq \mathcal{L}} z^P(L). \tag{1}$$

While it is possible to derive a mathematical programming formulation for **PSP-SL** by replacing the inner maximization by a minimization problem (that is, replacing $z^P(L)$ by $z^D(L)$), the constraints in $z^D(L)$) depend on the active learning opportunities in L. As a result, it appears "big-M" constraints together with corresponding binary variables are needed to account for the conditional nature of these constraints in the inner dual problem, which can make it hard to establish efficient solution procedures.

We observe, however, that it is possible to characterize optimal solutions to **PSP-SL** through a modified set of Bellman equations. Let variables u_j^0 and u_j^1 represent the completion time of job $j \in \mathcal{V}$ in the cases that j does not learn and that j learns, respectively. Then the recursive equations are as follows:

$$u_s = 0, \tag{2a}$$

$$u_j^0 = \max_{i:(i,j)\in\mathcal{A}} u_i + d_j, \qquad \forall j \in \mathcal{V}, \tag{2b}$$

$$u_j^1 = \max(u_j^0 - l_j, u_i + d_j - l_j), \qquad \forall (i,j) \in \mathcal{L}, \tag{2c}$$

$$u_j = \begin{cases} u_j^0, & j \notin \mathcal{V}(\mathcal{L}); \\ \min(u_j^0, u_j^1), & j \in \mathcal{V}(\mathcal{L}). \end{cases}, \qquad \forall j \in \mathcal{V}. \tag{2d}$$

The following proposition shows that solutions satisfying these Bellman equations provide an optimal solution to **PSP-SL**.

PROPOSITION 1. *Let* $(u^*, u^{0,*}, u^{1,*})$ *be a solution that satisfies the Bellman conditions (2), and let*

$$L^* = \{(i,j) \in \mathcal{L} : u_j^{1,*} < u_j^{0,*}\}.$$

Then, L^* *is an optimal solution to* ***PSP-SL****.*

Proof. As a first step, we observe that $\mathcal{A} \cup L^*$ is acyclic. To understand this, suppose that $\mathcal{A} \cup L^*$ includes a directed cycle $\mathcal{C} \subseteq \mathcal{A} \cup L^*$. We can partition the arcs in this cycle into two sets: $\mathcal{C}^0 = \{(i,j) \in \mathcal{C} : j \notin \mathcal{V}(L^*)\}$ and $\mathcal{C}^1 = \{(i,j) \in \mathcal{C} : j \in \mathcal{V}(L^*)\}$. For an arc $(i,j) \in \mathcal{C}^0$ we

have $u_j \geqslant u_i + d_j$ and for an arc $(i,j) \in \mathcal{C}^1$ we have $u_j \geqslant u_i + d_j - l_j$. Then, the aggregation of these inequalities gives

$$\sum_{j \in \mathcal{V}(\mathcal{C})} u_j \geqslant \sum_{i \in \mathcal{V}(\mathcal{C})} u_i + \sum_{j \in \mathcal{V}(\mathcal{C})} d_j - \sum_{j \in \mathcal{V}(\mathcal{C}) \cap \mathcal{V}(L^*)} l_j,$$

which can be rewritten as

$$\sum_{j \in \mathcal{V}(\mathcal{C}) \cap \mathcal{V}(L^*)} l_j \geqslant \sum_{j \in \mathcal{V}(\mathcal{C})} d_j.$$

This yields a contradiction because we assume $d_j > l_j$ for all $j \in \mathcal{V}(\mathcal{L})$.

Next, we show that the Bellman equations (2) are sufficient to solve **PSP-SL**, that is,

$$u_t^* = \min_{L \subseteq \mathcal{L}} z^P(L).$$

Suppose there exists some $\hat{L} \subseteq \mathcal{L}$ such that $z^P(\hat{L}) < u_t^*$ and let $\hat{u}$ be an optimal solution to $z^D(\hat{L})$. We show that $u_j^* \leqslant \hat{u}_j$ holds for all $j \in \mathcal{V}$ by induction. Because $\mathcal{A} \cup \hat{L}$ is acyclical, we first determine a topological ordering $\tau = \text{TopoSort}(\mathcal{V}, \mathcal{A} \cup \hat{L})$ of the nodes in $\mathcal{V}$. We observe that $u_0^* = \hat{u}_{\tau(1)} = 0$ (assuming that $s = 0$). Now, consider some $k > 0$ and suppose that $u_{\tau(k')}^* \leqslant \hat{u}_{\tau(k')}$ for all $k' < k$. We can separate two cases in which the activity $\tau(k)$ is either a learner or not in $\hat{L}$:

Case 1: $\tau(k) \notin \mathcal{V}(\hat{L})$.

In this case, we have $\hat{u}_{\tau(k)} = \max_{(i,\tau(k)) \in \mathcal{A}} \hat{u}_i + d_{\tau(k)}$. However,

$$u_{\tau(k)}^* = \min(u_{\tau(k)}^{0,*}, u_{\tau(k)}^{1,*}) \leqslant u_{\tau(k)}^{0,*} = \max_{(i,\tau(k)) \in \mathcal{A}} u_i^* + d_{\tau(k)}.$$

Given that $u_i^* \leqslant \hat{u}_i$ for all i such that $(i,\tau(k)) \in \mathcal{A}$, we have that $u_{\tau(k)}^* \leqslant \hat{u}_{\tau(k)}$.

Case 2: $\tau(k) \in \mathcal{V}(\hat{L})$.

In this case, we have $\hat{u}_{\tau(k)} = \max_{(i,\tau(k)) \in \mathcal{A} \cup L} \hat{u}_i + d_{\tau(k)} - l_{\tau(k)}$. However,

$$u_{\tau(k)}^* = \min(u_{\tau(k)}^{0,*}, u_{\tau(k)}^{1,*}) \leqslant u_{\tau(k)}^{1,*} = \max_{(i,\tau(k)) \in \mathcal{A} \cup L} u_i^* + d_{\tau(k)} - l_{\tau(k)}.$$

Again we have $u_{\tau(k)}^* \leqslant \hat{u}_{\tau(k)}$, given that $u_i^* \leqslant \hat{u}_i$ for all i such that $(i,\tau(k)) \in \mathcal{A} \cup \hat{L}$.

This implies that $z^* = u_t^* \leqslant \hat{u}_t = z^P(\hat{L})$, which completes the proof. ■

As a result, solving **PSP-SL** amounts to determining a solution to these modified Bellman equations. Observe that these general optimality conditions do not rely on topological sorting since no topological ordering exists for cyclic graphs $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$. Furthermore, the max-min nature of equations (2) significantly complicates the problem and it is not obvious how to efficiently formulate it as a linear program. In the remainder of this section, we propose an efficient iterative approach to find a solution to these modified Bellman equations.

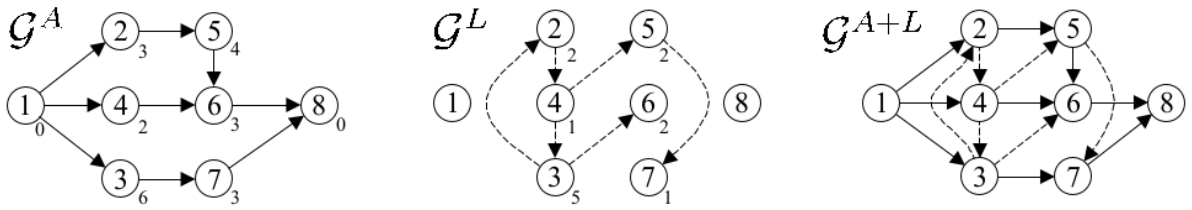

Figure 1 **Precedence network $\mathcal{G}^{\mathcal{A}}$ (left), learning network $\mathcal{G}^{\mathcal{L}}$ (center), and combined network $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ (right) for an example project with 8 activities, 9 precedences (solid arcs), and 6 learning opportunities (dashed arcs).**

4.1. An Example Project

To describe our approach and the key algorithmic techniques behind it, we will use the following example project. Consider an instance of **PSP-SL** with activity set $\mathcal{V} = \{s = 1, .., 8 = t\}$. The precedence network $\mathcal{G}^{\mathcal{A}}$, the learning network $\mathcal{G}^{\mathcal{L}}$ and the combined network $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ are as shown in Figure 1. Activity durations (d_j) and learning effects (l_j) are displayed beside the corresponding activity nodes of $\mathcal{G}^{\mathcal{A}}$ and $\mathcal{G}^{\mathcal{L}}$, respectively. We emphasize that $\mathcal{G}^{\mathcal{L}}$ is not acyclic due to its unique directed cycle $(2-4-3-2)$, and, therefore, also $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is not.

An optimal schedule for the corresponding PSP (without learning opportunities; i.e., $\mathcal{L} = \varnothing$) is given in Figure 2 (left). The unique longest (critical) path $(1-2-5-6-8)$ yields a project makespan of 10 time periods. When considering the learning opportunities from Figure 1 in this instance, the optimal makespan reduces to 6 time periods which we will prove later. The corresponding schedule is shown in Figure 2 (right). Only 3 of the 6 learning opportunities are active and the optimal set of active learning opportunities equals $L^* = \{(3,6),(4,3),(4,5)\}$.

We emphasize that this example aims to illustrate the learning mechanism, rather than a specific application. As an example application, one could consider planning for an apartment building construction project. Activities would represent tasks such as flooring, plumbing, and paint jobs, learning opportunities would naturally be obtained when workers get acquainted with the specifics of the building such as floor plans, materials used and the chosen finishing.

4.2. Optimization Strategies

In this section, we develop efficient algorithms for **PSP-SL**. We first explain the challenges connected to the presence of cycles in learning networks. To understand the complications

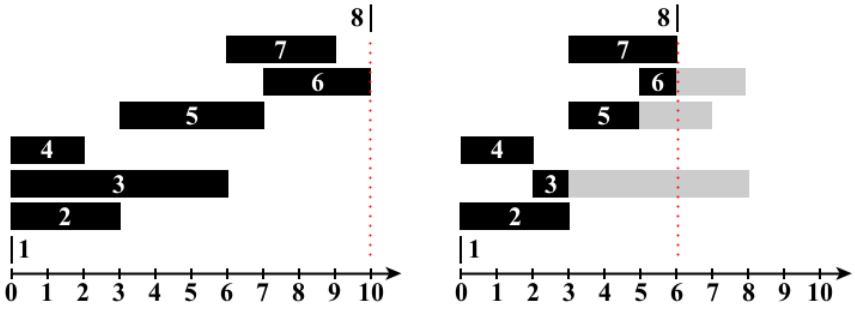

Figure 2 **The optimal project schedule for the PSP without learning opportunities (left), and when considering learning opportunities in the PSP-SL (right).**

these cycles introduce, we start by considering single-pass and multi-pass methods that might result in a sub-optimal solution. Afterwards, we extend these approaches and show that an iterative strategy can be used to solve the problem to optimality in polynomial time.

4.2.1. Topological Ordering for the Acyclic Case As a first step, we consider the special case of **PSP-SL** where $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is acyclic. Given a set of learning opportunities $\mathcal{L}$, we can use Algorithm 1 to derive an optimal schedule based on a topological ordering of the nodes in $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$.

Algorithm 1 TopoSortOpt$(\mathcal{V}, \mathcal{A}, \mathcal{L})$

$\tau \leftarrow \text{TopoSort}(\mathcal{V}, \mathcal{A} \cup \mathcal{L})$ ▷ Compute topological ordering of $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$

$u_{\tau(1)} \leftarrow 0$ ▷ Schedule first activity in ordering $(\tau(1) = s)$

for $k \leftarrow 2, \ldots, |\mathcal{V}|$ **do** ▷ Iteratively schedule subsequent activities

$u^0_{\tau(k)} \leftarrow \max_{(i,\tau(k))\in\mathcal{A}} u_i + d_{\tau(k)}$

$u^1_{\tau(k)} \leftarrow \max_{(i,\tau(k))\in\mathcal{A}\cup\mathcal{L}} u_i + d_{\tau(k)} - l_{\tau(k)}$

$$u_{\tau(k)} \leftarrow \begin{cases} u^0_{\tau(k)}, & \tau(k) \notin \mathcal{V}(\mathcal{L}) \\ \min(u^0_{\tau(k)}, u^1_{\tau(k)}), & \tau(k) \in \mathcal{V}(\mathcal{L}) \end{cases}$$

$L \leftarrow \{(i,j) \in \mathcal{L} : u^1_j < u^0_j\}$ ▷ Identify active learning opportunities

return (u, L) ▷ Return schedule and active learning opportunities

LEMMA 1. *Algorithm 1 (TopoSortOpt) computes an optimal project schedule for* ***PSP-SL*** *in* $\mathcal{O}(|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|)$ *if the digraph* $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ *is acyclic.*

Proof. Optimality follows because the schedule produced by algorithm will satisfy the modified Bellman equations (2) by construction, while the complexity of the topological sorting procedure is well-known (see, for example, Ahuja et al. 1988). ■

Algorithm 1 uses the modified Bellman equations (2) to generalize the well-known exact method for the PSP described in Section 4 and returns both a vector u of optimal activity completion times and a set of active learning opportunities L.

In general, however, $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ can be cyclic even if $\mathcal{G}^{\mathcal{L}}$ is acyclic. To see this, consider the example project in Figure 1 with a modified set of learning opportunities $\mathcal{L} = \{(7,4),(6,7)\}$. Because all nodes in $\mathcal{G}^{\mathcal{L}}$ have an indegree of at most one the structure of $\mathcal{G}^{\mathcal{L}}$ can be characterized as a directed pseudo-forest, that is, the disjoint union of pseudo-arborescences. A pseudo-arborescence is an arborescence with at most one additional arc forming a directed cycle. The structure of $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$, however, is more complicated and – to the best of our knowledge – does not appear to have an intuitive structural characterization.

The TopoSortOpt method (Algorithm 1) highlights that the key challenge in solving **PSP-SL** arises when the learning network $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ contains cycles. Algorithm 1 cannot be used for the cyclic case since no topological ordering of $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ exists. However, every possible reduction of the set of learning opportunities that eliminates all cycles might be used to calculate an approximate schedule using TopoSortOpt. When we consider the instance outlined in Section 4.1, for example, $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ becomes acyclic when removing learning opportunity $(3,2)$, and Algorithm 1 returns the optimal schedule with makespan 6, depicted in Figure 2 (right). Eliminating the unique cycle by removing $(2,4)$ or $(4,3)$ on the other hand instead results in a larger makespan. In general, there could be a large number of cycles and an exponential number of such possible reductions. Therefore, it is not clear a-priori which acyclic subgraph of $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ should be considered.

4.2.2. Organic Learning To establish efficient procedures for determining an optimal set of active learning opportunities, we first observe that learning can occur without its consideration during the actual scheduling phase. Beneficial learning opportunities can be identified in an a posteriori fashion. In this section, we explore approaches that take advantage of such *organic learning* in an iterative way. To this end, we show that selected learning opportunities can be enforced if respecting the acyclic property. At the same time, these approaches highlight the inherent limitations of organic learning and illustrate the challenges that arise in project scheduling problems that involve learning.

PROPOSITION 2. *Let $\mathcal{L}' \subseteq \mathcal{L}$ be a set of learning opportunities such that $\mathcal{G}^{\mathcal{A}+\mathcal{L}'}$ is acyclic. Then, the corresponding **PSP-SL** instance with the additional requirement that precisely all learning opportunities in $\mathcal{L}'$ are active can be solved to optimality efficiently.*

Proof. To see this, consider the auxiliary **PSP-SL** instance using activities in $\mathcal{V}$, the extended precedence set $\mathcal{A}' := \mathcal{A} \cup \mathcal{L}'$, no learning opportunities ($\mathcal{L} := \varnothing$), and a reduced duration $d_j := d_j - l_j$ for each activity $j \in \mathcal{V}(\mathcal{L}')$. Then, Algorithm 1 (without explicit learning opportunities) can be applied to obtain an optimal schedule where all activities in $\mathcal{V}(\mathcal{L}')$ learn. The proposition follows from $\mathcal{G}^{\mathcal{A}'}$ being acyclic by assumption and the fact that every learning opportunity is enforced by construction of the auxiliary problem through corresponding additional precedences. ■

To understand the process of organic learning, consider the optimal schedule for the example (PSP) shown in Figure 2 (left). We argue that activity 5 learns organically from activity 4 since the necessary precedence requirement is naturally met (i.e., $u_4 = 2 \leqslant 3 = u_5 - d_5$). Similarly, organic learning also occurs between activities 3 and 6. Using Proposition 2, we can enforce both learning opportunities and re-optimize the schedule using Algorithm 1 for the PSP. The result is an improved schedule with a makespan of 9 as shown in Figure 3 (left). We refer to this organic learning-based method as OrgaOpt.

The schedule found by OrgaOpt can be further improved by considering organic learning opportunities in an iterative fashion. The organic learning opportunity $(5, 7)$ can only be identified after the initial re-optimization using OrgaOpt. Pursuing another iteration of OrgaOpt where activity 7 also learns organically (in addition to activities 5 and 6) results in an improved makespan of 8, as shown in Figure 3 (right). This mechanism can be repeated until no organic learning can be identified.

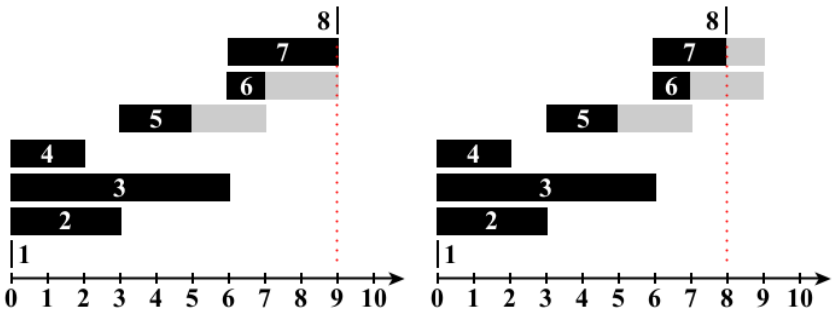

Figure 3 **The project schedule for the PSP-SL instance obtained after single-step organic learning OrgaOpt (left), and after iterative organic learning OrgaOptIter (right).**

Algorithm 2 OrgaOptIter($\mathcal{V}, \mathcal{A}, \mathcal{L}$)

$k \leftarrow 0,\ A \leftarrow \varnothing$
repeat ▷ Iteratively enforce organic learning opportunities
  $k \leftarrow k + 1$
  $(u_k, L) \leftarrow TopoSortOpt(\mathcal{V}, \mathcal{A} \cup A, \varnothing)$ ▷ Re-optimize PSP
  $L_k \leftarrow \{(i,j) \in \mathcal{L} \backslash A : u_{k,j} \geqslant u_{k,i} + d_j\}$ ▷ Identify organic learning opportunities
  for all $(i,j) \in L_k$ **do**
    $d_j \leftarrow d_j - l_j$ ▷ Update organic learners' durations
  $A \leftarrow A \cup L_k$ ▷ Update enforced learning opportunities
until $L_k = \varnothing$ ▷ Stop when no organic learning found
return (u_k, A) ▷ Return schedule and active learning opportunities

We refer to the resulting iterative organic learning procedure as OrgaOptIter, which is formally described in Algorithm 2. The method repeatedly executes the TopoSortOpt algorithm without learning opportunities which corresponds to solving a PSP after updating the precedences using Proposition 2. Note that u_k denotes the schedule computed in iteration k and $u_{k,j}$ is the corresponding completion time of activity j. We explicitly track the iteration numbers to ease exposition in subsequent proofs.

In addition, we establish the complexity of OrgaOptIter in Lemma 2.

Lemma 2. *Algorithm OrgaOptIter returns a feasible schedule for* ***PSP-SL*** *in* $\mathcal{O}((|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|) \cdot |\mathcal{L}|)$.

Proof. By induction, $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is acyclic in each iteration by definition of $\mathcal{A}$ and A. Therefore, u_k always corresponds to a feasible schedule using Lemma 1. The cardinality of the set of additional precedences A strictly increases in each iteration until no new organic learning opportunities are identified. The maximum number of iterations is $|\mathcal{L}|$ since A is extended by arcs from $\mathcal{L}$, i.e., $k \leqslant |\mathcal{L}|$ holds. ■

Corollary 1. *OrgaOpt returns a feasible schedule for* ***PSP-SL*** *in* $\mathcal{O}(|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|)$.

Proof. OrgaOpt is a special case of OrgaOptIter obtained by limiting the number of iterations to two. ■

Both the OrgaOpt and OrgaOptIter algorithms can be applied when cycles are present in $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$. These methods, however, do not necessarily converge to a global optimum –

even when $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is acyclic. To see this, it suffices to consider the example project from Section 4.1 without learning opportunity $(3, 2)$. Nevertheless, we recall that this acyclic case can be solved optimally by Algorithm 1.

4.2.3. Achieving Optimality The suboptimality of the OrgaOptIter algorithm can be attributed to the fact that it is an additive procedure, that is, learning opportunities can only be added in each iteration. To overcome this, we next consider a more general approach that also allows previously active learning opportunities to be removed. Specifically, we modify Algorithm 2 to allow the deactivation of learning opportunities in line 4 of the algorithm.

The key idea behind this approach is to consider a set of *candidate* learning opportunities in each iteration of the procedure that generalizes the organic learning opportunities identified in the OrgaOptIter algorithm. Given a schedule u, a non-active learning opportunity $(i, j) \in \mathcal{L}$ is a candidate if

$$u_j > u_i + d_j - l_j,$$

that is, adding the learning opportunity has the potential to reduce the completion time of activity j. After adding such candidate learning opportunities, the TopoSortOpt algorithm can then be used to determine which of the learning opportunities should be activated. However, we require that candidate learning opportunities do not introduce cycles in the graph $\mathcal{G}^{\mathcal{A}+L}$. This is established in the following lemma.

LEMMA 3. *Let u be an optimal project schedule with the set of active learning opportunities L, and define the candidate set*

$$C = \{(i, j) \in \mathcal{L} : u_j > u_i + d_j - l_j\}.$$

Then, $\mathcal{G}^{\mathcal{A}+L+C}$ is acyclic.

We omit a formal proof of the lemma as it follows the reasoning outlined in the first step of the proof of Proposition 1, using $L \cup C$ in lieu of L^*. Algorithm 3 describes the resulting OptIter procedure.

Algorithm 3 OptIter$(\mathcal{V}, \mathcal{A}, \mathcal{L})$

$k \leftarrow 0,\ L_0 \leftarrow \varnothing,\ C_0 \leftarrow \varnothing$
repeat ▷ Iteratively integrate learning opportunities
  $k \leftarrow k + 1$
  $(u_k, L_k) \leftarrow TopoSortOpt(\mathcal{V}, \mathcal{A}, L_{k-1} \cup C_{k-1})$ ▷ Re-optimize acyclic PSP-SL
  $C_k \leftarrow \{(i,j) \in \mathcal{L} : u_{k,j} > u_{k,i} + d_j - l_j\}$ ▷ Identify candidates
until $C_k = \varnothing$ ▷ Stop when no candidates found
return (u_k, L_k) ▷ Return schedule and active learning opportunities

Algorithm 3 generalizes organic learning for solving arbitrary **PSP-SL** instances, as it guarantees an optimal solution and has polynomial time complexity. We first show that the algorithm generates an optimal solution upon termination.

THEOREM 1. *Upon termination, Algorithm 3 (OptIter) returns an optimal schedule for the PSP-SL.*

Proof. To establish optimality, we show that the final solution (u, L) satisfies the modified Bellman equations (2). By construction, we have

$$u_j = \begin{cases} \max_{(i,j)\in\mathcal{A}} u_i + d_j, & \text{if } j \notin \mathcal{V}(L); \\ \max_{(i,j)\in\mathcal{A}\cup L} u_i + d_j - l_j, & \text{if } j \in \mathcal{V}(L). \end{cases} \quad \forall j \in \mathcal{V},$$

and can define

$$u_j^0 = \max_{i:(i,j)\in\mathcal{A}} u_i + d_j, \qquad \forall j \in \mathcal{V},$$

$$u_j^1 = \max(u_j^0 - l_j, u_i + d_j - l_j), \quad \forall (i,j) \in \mathcal{L}.$$

Observe that $u_j = u_j^0$ for $j \notin \mathcal{V}(L)$, and $u_j = u_j^1$ for $j \in \mathcal{V}(L)$. To ensure that the optimality equations are met, we need to show that

$$u_j = \begin{cases} u_j^0, & j \notin \mathcal{V}(\mathcal{L}); \\ \min(u_j^0, u_j^1), & j \in \mathcal{V}(\mathcal{L}). \end{cases}, \qquad \forall j \in \mathcal{V}.$$

Clearly this holds for $j \notin \mathcal{V}(\mathcal{L})$. For an activity $j \in \mathcal{V}(L)$, we know that

$$u_j^1 = u_j \leqslant \max_{(i,j)\in\mathcal{A}} u_i + d_j = u_j^0.$$

Thus, $u_j = u_j^1 = \min(u_j^0, u_j^1)$ and therefore the optimality equations again hold. Finally, for an activity $j \in \mathcal{V}(\mathcal{L}) \backslash \mathcal{V}(L)$ we have

$$u_j^0 = u_j = \max_{(i,j)\in\mathcal{A}} u_i + d_j.$$

When the algorithm terminates, there are no candidate learning opportunities and therefore $u_j \leqslant u_i + d_j - l_j$ for the learning opportunity $(i,j) \in \mathcal{L} \backslash L$, Thus, $u_j^0 \leqslant u_j^1$ and therefore the optimality equations also hold in this case. ■

Moreover, it is easy to see that the algorithm will terminate in a finite number of iterations. The addition of candidate learning opportunities identified in line 5 of the algorithm can only lead to a decrease of the completion times u_j $(j \in \mathcal{V})$ in each iteration, while (by induction on the topological ordering used by the TopoSortOpt algorithm in line 4) the completion time will strictly decrease for at least one activity. The challenge in establishing *polynomial* time convergence, however, lies in the fact that candidate learning opportunities might become active, active learning opportunities might become inactive, while inactive learning opportunities might become candidates. To demonstrate polynomial time convergence, we limit the number of such occurrences. Specifically, we show that the completion time of at least one additional activity will be fixed after each iteration and not decrease further in any future iterations. We establish polynomial time convergence of the OptIter algorithm in Theorem 1.

THEOREM 2. *Algorithm 3 (OptIter) computes an optimal schedule for the PSP-SL in* $\mathcal{O}((|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|) \cdot |\mathcal{V}|)$.

Proof. We show that the OptIter algorithm will terminate in at most $|\mathcal{V}|$ iterations. Given that each iteration requires $\mathcal{O}(|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|)$ time, this will establish the desired result.

Specifically, we show that in each iteration there is at least one activity whose completion time will become fixed. To that end, consider an iteration k and define

$$\underline{u}_k = \min_{(i,j)\in C_k} u_{k,i},$$

that is, $\underline{u}_k$ represents the earliest candidate teacher completion time in iteration k. We also define $\mathcal{F}_k = \{j \in \mathcal{V} : u_{k,j} = \underline{u}_k\}$ and $\mathcal{F} = \{j \in \mathcal{V} : u_{k,j} \leqslant \underline{u}_k\}$. Observe that $|\mathcal{F}_k| \geqslant 1$ by construction unless $C_k = \varnothing$ and the procedure terminates.

After completing $TopoSortOpt(\mathcal{V}, \mathcal{A}, L_k \cup C_k)$ in iteration k, we can observe that the completion times of the activities in $\mathcal{F}$ remain unchanged, that is, $u_{k+1,j} = u_{k,j}$ for $j \in \mathcal{F}$. In addition, we can also observe that any activity $j \in \mathcal{F}$ will neither be teacher nor a learner in any candidate learning opportunity during iteration $k+1$. Together, these observations guarantee that the size of $\mathcal{F}$ will strictly increase after each iteration of the procedure.

To see why the completion times of the activities in $\mathcal{F}$ remain unchanged after completing $TopoSortOpt(\mathcal{V}, \mathcal{A}, L_k \cup C_k)$, we note that executing $TopoSortOpt(\mathcal{V}, \mathcal{A}, L_k \cup C_k)$ can only impact the completion times of the learner activities in C_k and their successors in the acyclic graph $\mathcal{G}^{\mathcal{A} \cup L_k \cup C_k}$. Therefore, the completion times of the learner activities that are updated will always be larger than $\underline{u}_k$, and the same also holds for their successors.

By contradiction, we can show that an activity $j \in \mathcal{F}$ will neither be learner nor a teacher in any candidate learning opportunity during iteration $k+1$. First, suppose that $(i,j) \in C_{k+1}$, that is, j is a learner in iteration $k+1$. However, we have $i \notin \mathcal{F}$ for otherwise (i,j) would have been a candidate learning opportunity during iteration k. At the same time, $i \notin \mathcal{V} \backslash \mathcal{F}$ because $u_{k+1,j} \leqslant \underline{u}_k$ and $u_{k+1,i} > \underline{u}_k$ for all $i \notin \mathcal{V} \backslash \mathcal{F}$. Next, suppose that $(j,i) \in C_{k+1}$, that is, j is a teacher in iteration $k+1$. Because $u_{i,k+1} > u_{j,k+1} + d_i + l_i$ and activity completion times only decrease, this would imply that $(j,i) \in C_k$. However, after the execution of $TopoSortOpt(\mathcal{V}, \mathcal{A}, L_k \cup C_k)$ we have that $u_{i,k+1} \leqslant u_{j,k+1} + d_i + l_i$ for all $(j,i) \in C_k$. ■

Using the OptIter algorithm, the example project introduced in Section 4.1 can be solved to optimality in two iterations. After computing the initial schedule for the PSP (see Figure 2 (left)), it identifies the following learning opportunities: $C_1 = \{(4,3),(4,5),(3,6)\}$. The solution of the resulting acyclic PSP-L subproblem found by TopoSortOpt is given by $u_2 = (0,3,3,2,5,6,6)$ and has makespan 6.

To see that the OptIter algorithm might in fact deactivate learning opportunities, consider the 7-job example project illustrated in Figure 4. Note that $\mathcal{G}^{\mathcal{L}}$, and therewith $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$, is cyclic. After obtaining the optimal PSP schedule with makespan 9 in the first iteration, candidate learning opportunities $(5,4)$ and $(2,6)$ are considered. The resolution of the corresponding acyclic PSP-SL results in an improved makespan of 8, and opportunity $(4,3)$ becomes an additional candidate. Activity 3 does learn in the optimal schedule found in the third iteration. However, the previously useful learning of activity 6 is deactivated.

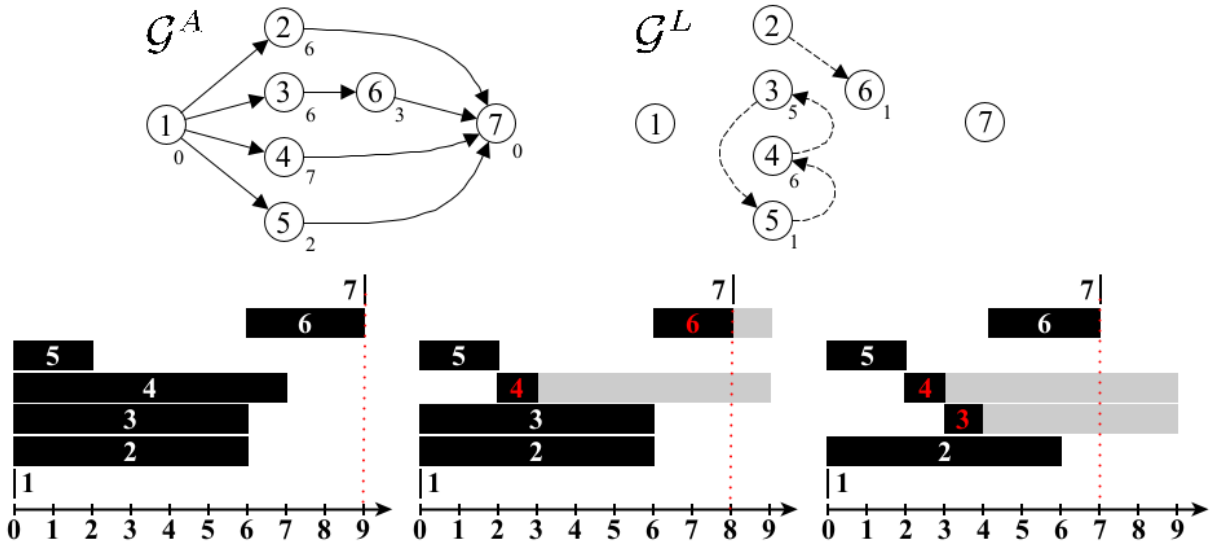

Figure 4 **An example project for which our optimal strategy OptIter deactivates learning opportunity** $(2,6)$ **in the third iteration, yielding the optimal schedule.**

5. Project Scheduling with Multiple Learning Opportunities

In this section we extend our model to incorporate multiple learning opportunities for each activity, and show that the OptIter algorithm can be modified to also find optimal solution in this general case.

As a first step, we again establish a formal problem statement by modifying the dual formulation to find the longest path and minimize makespan for a given set $L \subseteq \mathcal{L}$ of active learning opportunities. This yields the following formulation.

$$z^D(L) = \min\ u_t - u_s$$

$$\text{s.t. } u_j - u_i \geqslant d_j - l_j(L), \qquad \forall (i,j) \in \mathcal{A} \cup L.$$

Its corresponding primal formulation equals

$$z^P(L) = \max \sum_{(i,j)\in\mathcal{A}\cup L} [d_j - l_j(L)]\, x_{ij}$$

$$\text{s.t.} \sum_{i:(i,j)\in\mathcal{A}\cup L} x_{ij} - \sum_{i:(j,i)\in\mathcal{A}\cup L} x_{ij} = \begin{cases} -1, & j = s; \\ 1, & j = t; \\ 0, & \text{otherwise}, \end{cases} \qquad \forall j \in \mathcal{V},$$

$$x_{ij} \geqslant 0 \qquad \forall (i,j) \in \mathcal{A} \cup L.$$

Observe that the dual formulation will be infeasible if $\mathcal{A} \cup L$ contains a cycle (assuming that $d_j - l_j(L) > 0$ for all $j \in \mathcal{V}$); in that case, the primal is unbounded and we have $z^P(L) = \infty$.

The overall problem is to determine a set of learning opportunities such that the longest path is minimized. We can formulate this problem with multiple learning opportunities as

$$\textbf{PSP-ML}: \quad z = \min_{L \subseteq \mathcal{L}} z^P(L).$$

Following the same approach as for the single learning opportunity case, we first establish certain properties of optimal solutions to **PSP-ML**. Again, we use u_j to represent the earliest completion time of activity j after incorporating learning opportunities and aim to express those using a set of optimality equations. The optimality equations we define are:

$$u_s = 0, \tag{3a}$$

$$u_j = \min_{L_j \subseteq \mathcal{L}_j} \max_{(i,j) \in \mathcal{A} \cup L_j} u_i + d_j - l_j(L_j), \qquad \forall j \in \mathcal{V}. \tag{3b}$$

The makespan will correspond to the largest of these completion times, that is, $z = u_t$.

Next, we generalize Proposition 1 to show that solutions satisfying these new optimality equations provide an optimal solution to **PSP-ML**.

PROPOSITION 3. *Let u^* be a solution that satisfies the Bellman conditions (3), and let*

$$L_j^* \in \arg \min_{L_j \subseteq \mathcal{L}_j} \max_{(i,j) \in \mathcal{A} \cup L_j} u_i + d_j - l_j(L_j), \qquad \forall j \in \mathcal{V}.$$

*Then, $L^* = \bigcup_{j \in V} L_j^*$ is an optimal solution to **PSP-ML**.*

We omit the proof of Proposition 3 as it relies on the same reasoning as Proposition 1, in that we first establish that $\mathcal{A} \cup L^*$ is acyclic and use this to establish optimality by induction.

A key complication in this case is that the optimality equations require optimization over all *subsets* of learning opportunities for a given activity $j \in \mathcal{V}$. However, the following lemma shows that only a small number of such subsets need to be considered.

LEMMA 4. *Suppose $(i,j) \in L_j^*$ for some $j \in \mathcal{V}$, and let $L_{i,j}(u^*) = \{(i',j) \in \mathcal{L}_j : u_{i'}^* \leqslant u_i^*\}$. Then, without loss of optimality we can assume that $L_{i,j}(u^*) \subseteq L_j^*$.*

Proof. This follows because

$$\max_{(i,j)\in\mathcal{A}\cup L_j^*\cup L_{i,j}(u^*)} u_i^* + d_j - l_j(L_j^* \cup L_{i,j}(u^*)) \leqslant \max_{(i,j)\in\mathcal{A}\cup L_j^*} u_i^* + d_j - l_j(L_j^*),$$

since

$$\max_{(i,j)\in\mathcal{A}\cup L_j^*\cup L_{i,j}(u^*)} u_i^* = \max_{(i,j)\in\mathcal{A}\cup L_j^*} u_i^*$$

following the definition of of $L_{i,j}$, and

$$l_j(L_j^* \cup L_{i,j}(u^*)) \leqslant l_j(L_j^*),$$

due to the monotonicity of l_j. ■

Lemma 4 implies that the optimization over all *subsets* of learning opportunities can be achieved by optimizing over all *individual* learning opportunities (i,j) for a given learner activity $j \in \mathcal{V}$ by considering the subset of learning opportunities whose corresponding teacher activities have completion times earlier that activity i. We formalize this in the following corollary.

Corollary 2. *Let u^* be a solution that satisfies the Bellman conditions (3), and let*

$$u_j^{0,*} = \max_{i:(i,j)\in\mathcal{A}} u_i + d_j, \qquad \forall j \in \mathcal{V},$$

$$u_j^{1,*} = \min_{(i,j)\in\mathcal{L}_j} \max_{(i,j)\in\mathcal{A}\cup L_{i,j}(u^*)} u_i^* + d_j - l_j(L_{i,j}(u^*)), \qquad \forall j \in \mathcal{V}.$$

Then,

$$u_j^* = \min(u_j^{0,*}, u_j^{1,*}), \qquad \forall j \in \mathcal{V}.$$

5.1. Optimization Strategies: The Acyclic Case

We first adapt Algorithm 1 to handle multiple learning opportunities in the **PSP-ML** where $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is acyclic. Given a set of learning opportunities $\mathcal{L}$, we can now use Algorithm 4 to derive an optimal schedule based on a topological ordering of the nodes in $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$.

Lemma 5. *Algorithm 4 (TopoSortOptGen) computes an optimal project schedule for **PSP-ML** in $\mathcal{O}(|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|(1 + \log(|\mathcal{L}|)))$ if the digraph $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is acyclic.*

Proof. Optimality follows because the schedule produced by algorithm will satisfy the modified Bellman equations (3) by construction, while the complexity follows from the complexity of the topological sorting procedure together with the complexity of sorting the learning opportunities. ■

Algorithm 4 TopoSortOptGen$(\mathcal{V}, \mathcal{A}, \mathcal{L})$

$\tau \leftarrow \text{TopoSort}(\mathcal{V}, \mathcal{A} \cup \mathcal{L})$ ▷ Compute topological ordering of $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$

$u_{\tau(1)} \leftarrow 0$ ▷ Schedule first activity in ordering $(\tau(1) = s)$

for $k \leftarrow 2, \ldots, |\mathcal{V}|$ **do** ▷ Iteratively schedule subsequent activities

$u^0_{\tau(k)} \leftarrow \max_{(i,\tau(k))\in\mathcal{A}} u_i + d_{\tau(k)}$

$u^1_{\tau(k)} \leftarrow \min_{(i,\tau(k))\in\mathcal{L}_j} \max_{(i,\tau(k))\in\mathcal{A}\cup L_{i,j}(u)} u_i + d_{\tau(k)} - l_{\tau(k)}(L_{i,\tau(k)}(u))$

$(i^1, \tau(k)) \leftarrow \arg\min_{(i,\tau(k))\in\mathcal{L}_j} \max_{(i,\tau(k))\in\mathcal{A}\cup L_{i,j}(u)} u_i + d_{\tau(k)} - l_{\tau(k)}(L_{i,\tau(k)}(u))$

$u_{\tau(k)} \leftarrow \min(u^0_{\tau(k)}, u^1_{\tau(k)})$,

$$L_{\tau(k)} \leftarrow \begin{cases} \varnothing, & u^0_{\tau(k)} \leqslant u^1_{\tau(k)} \\ L_{i^1,\tau(k)}(u), & \text{otherwise} \end{cases}$$

$L \leftarrow \bigcup_{j\in\mathcal{V}} L_j$ ▷ Identify active learning opportunities

return (u, L) ▷ Return schedule and active learning opportunities

5.2. Optimization Strategies: The General Case

To establish a polynomial-time algorithm to solve **PSP-ML** in the general case we again rely on identifying candidate learning opportunities, using the observations in our proof of Lemma 5. Given a schedule u and some non-active learning opportunity $(i, j) \in \mathcal{L}_j$, we use $L_{i,j}(u) = \{(i', j) \in \mathcal{L}_j : u_{i'} \leqslant u_i\}$ and observe that the learning opportunities in $L_{i,j}$ provide a candidate set if

$$u_j > u_i + d_j - l_j(L_{i,j}(u)).$$

Because a candidate set $L_{i,j}(u)$ includes all learning opportunities whose teacher has a completion time that is at most u_i, it suffices to only consider the non-active learning opportunity $(i, j) \in \mathcal{L}_j$ with the highest teacher completion time u_i. More precisely, we can define

$$\overline{u}_j = \max_{(i,j)\in\mathcal{L}_j : u_j > u_i + d_j - l_j(L_{i,j}(u))} u_i$$

and its corresponding candidate set

$$C_j = \{(i, j) \in \mathcal{L}_j : u_i \leqslant \overline{u}_j\}.$$

Adding such candidate sets has the potential to reduce the completion time of activity j. As before, however, the following lemma first establishes that adding these learning opportunities does not introduce cycles in the graph $\mathcal{G}^{\mathcal{A}+L}$.

LEMMA 6. *Let u be an optimal project schedule with the set of active learning opportunities L, let $\overline{u}_j = \max_{(i,j)\in\mathcal{L}_j : u_j > u_i + d_j - l_j(L_{i,j}(u))} u_i$, and let $C_j = \{(i,j) \in \mathcal{L}_j : u_i \leqslant \overline{u}_j\}$ for all $j \in \mathcal{V}$. Defining the candidate set*

$$C = \bigcup_{j\in\mathcal{V}} C_j,$$

we have that $\mathcal{G}^{\mathcal{A}+L+C}$ is acyclic.

We again omit a formal proof of the lemma as it is based on the reasoning outlined in the first step of the proof of Proposition 3. Algorithm 5 describes the resulting procedure for determining optimal project schedules in the general case.

Algorithm 5 OptIterGen($\mathcal{V}, \mathcal{A}, \mathcal{L}$)

$k \leftarrow 0,\ L_0 \leftarrow \varnothing,\ C_{0,j} \leftarrow \varnothing\ \forall j \in \mathcal{V}$
repeat ▷ Iteratively integrate learning opportunities
  $k \leftarrow k + 1$
  $(u_k, L_k) \leftarrow \mathit{TopoSortOptGen}(\mathcal{V}, \mathcal{A}, L_{k-1} \cup \bigcup_{j\in\mathcal{V}} C_{k-1,j})$ ▷ Re-optimize
  for all $j \in \mathcal{V}$ **do** ▷ Identify candidates
    $\overline{u}_{k,j} = \max_{(i,j)\in\mathcal{L}_j : u_j > u_i + d_j - l_j(L_{i,j}(u))} u_i$
    $C_{k,j} = \{(i,j) \in \mathcal{L}_j : u_i \leqslant \overline{u}_{k,j}\}$
until $\bigcup_{j\in\mathcal{V}} C_{k,j} = \varnothing$ ▷ Stop when no candidates found
return (u_k, L_k) ▷ Return schedule and active learning opportunities

We first show that Algorithm 5 generates an optimal solution upon termination.

THEOREM 3. *Upon termination, Algorithm 5 (OptIterGen) returns an optimal schedule for PSP-ML.*

Proof. To establish optimality, we show that the final solution (u, L) satisfies the modified Bellman equations (3). By construction, we have

$$\begin{aligned} u_s &= 0, \\ u_j &= \max_{(i,j)\in\mathcal{A}\cup\mathcal{L}_j} u_i + d_j - l_j(L \cap \mathcal{L}_j), \qquad \forall j \in \mathcal{V}. \end{aligned}$$

Suppose now there exists some $\hat{L}_j \subseteq \mathcal{L}_j$ for some $j \in \mathcal{V}$ such that

$$u_j > \max_{(i,j)\in\mathcal{A}\cup\mathcal{L}_j} u_i + d_j - l_j(\hat{L}_j \cap \mathcal{L}_j).$$

This implies that there exist some $(i,j) \in \hat{L}_j$ such that $u_j > u_i + d_j - l_j(L_{ij}(u))$, which would yield a contradiction because the candidate set is empty upon termination. ■

Finally, we establish polynomial time convergence of the OptIterGen algorithm in Theorem 4.

THEOREM 4. *Algorithm 5 (OptIterGen) computes an optimal schedule for PSP-ML in* $\mathcal{O}((|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|(1 + \log(|\mathcal{L}|))) \cdot |\mathcal{V}|)$ *time.*

Proof. We show that the OptIter algorithm will terminate in at most $|\mathcal{V}|$ iterations. Given that each iteration requires $\mathcal{O}(|\mathcal{V}| + |\mathcal{A}| + |\mathcal{L}|(1 + \log(|\mathcal{L}|)))$ time, this will establish the desired result.

As before, we show that in each iteration there is at least one activity whose completion time will become fixed. To that end, consider an iteration k and define

$$\underline{u}_k = \min_{j \in \mathcal{V}: C_{k,j} \neq \emptyset} \overline{u}_{k,j},$$

that is, $\underline{u}_k$ represents the earliest of the latest candidate teacher completion times in iteration k. We again define $\mathcal{F}_k = \{j \in \mathcal{V} : u_{k,j} = \underline{u}_k\}$ and $\mathcal{F} = \{j \in \mathcal{V} : u_{k,j} \leqslant \underline{u}_k\}$. Observe that $|\mathcal{F}_k| \geqslant 1$ by construction unless the candidate set is empty and the procedure terminates.

After completing $TopoSortOptGen(\mathcal{V}, \mathcal{A}, L_k \cup \bigcup_{j \in \mathcal{V}} C_{k,j})$ in iteration k, we can observe that the completion times of the activities in $\mathcal{F}$ remain unchanged, that is, $u_{k+1,j} = u_{k,j}$ for $j \in \mathcal{F}$. In addition, we can also observe that any activity $j \in \mathcal{F}$ will neither be teacher nor a learner in any candidate learning opportunity during iteration $k+1$. Together, these observations guarantee that the size of $\mathcal{F}$ will strictly increase after each iteration of the procedure.

To see why the completion times of the activities in $\mathcal{F}$ remain unchanged after completing $TopoSortOptGen(\mathcal{V}, \mathcal{A}, L_k \cup \bigcup_{j \in \mathcal{V}} C_{k,j})$, we note that executing $TopoSortOptGen(\mathcal{V}, \mathcal{A}, L_k \cup \bigcup_{j \in \mathcal{V}} C_{k,j})$ can only impact the completion times of the learner activities in $\bigcup_{j \in \mathcal{V}} C_{k,j}$ and their successors in the acyclic graph $\mathcal{G}^{\mathcal{A} \cup L_k \cup \bigcup_{j \in \mathcal{V}} C_{k,j}}$. Therefore, the completion times of the learner activities that are updated will always be larger than $\underline{u}_k$, and the same also holds for their successors.

By contradiction, we can show that an activity $j \in \mathcal{F}$ will neither be teacher nor a learner in any candidate learning opportunity during iteration $k+1$. First, suppose that $C_{k+1,j} = L_{i,j}(u_{k+1})$ for some $(i,j) \in \mathcal{L}_j$, that is, j is a learner in iteration $k+1$. Because the completion

times of activities in $\mathcal{F}$ do not change, we have $i \notin \mathcal{F}$ or otherwise we would have $L_{i,j}(u_{k+1}) \subseteq C_{k,j}$ during iteration k and after executing $TopoSortOptGen(\mathcal{V}, \mathcal{A}, L_k \cup \bigcup_{j\in\mathcal{V}} C_{k,j})$ we have $u_{k+1,j} \leqslant u_{k+1,i} + d_j - l_j(L_{i,j}(u_{k+1})$ for all $(i,j) \in C_{k,j}$. At the same time, $i \notin \mathcal{V}\backslash\mathcal{F}$ because $u_{k+1,j} \leqslant \underline{u}_k$ and $u_{k+1,i} > \underline{u}_k$ for all $i \notin \mathcal{V}\backslash\mathcal{F}$. Next, suppose that $L_{j,i}(u_{k+1}) \subseteq C_{k+1,i}$, that is, j is a teacher in iteration $k+1$ for some activity i. Because $u_{i,k+1} > u_{j,k+1} + d_i + l_i(L_{j,i}(u_{k+1}))$ and activity completion times only decrease, this would imply that $(j,i) \in C_{k,j}$. However, after the execution of $TopoSortOpt(\mathcal{V}, \mathcal{A}, L_k \cup \bigcup_{j\in\mathcal{V}} C_{k,j})$ we have that $u_{i,k+1} \leqslant u_{j,k+1} + d_i + l_i(L_{j,i}(u_k))$ for all $(j,i) \in C_{k,j}$. ■

6. Computational Study

We perform a computational analysis to quantify (1) makespan benefits, (2) learning opportunity utilization, and (3) algorithm performance for the discussed methods. In our experiments, we use a subset of 180 instances (Hill and Vossen 2024) based on PSPlib (Kolisch and Sprecher 1997) instances introduced in Hill et al. (2021) for the resource-constrained case. In these instances, learning opportunities are randomly incorporated respecting the assumptions (no free learning, no impossible learning, at most one teacher). We adapted these instances to our model by discarding the resource features. These PSP-SL test cases contain 30 or 120 activities (90+90 instances), from 20% to 80% learners, and duration reduction potentials of 10%, 50%, and 90%. For the PSP-ML, we used these instances as starting point to create instances for extended scenarios. We implemented the methods using the computer algebra system MAPLE[1] and ran the tests on an Intel Core i7-7600 2.80 GHz machine with 16 GB RAM.

6.1. Single Learning Opportunities

Tables 2 and 3 contain instance and solution details for 30-job instances and 120-job-instances, respectively. The average run time for 30-job instances is 0.3 seconds, and 6.1 seconds for the 120-job instances. The column data can be described as follows.

#: Instance number

$|\mathcal{V}|$: Number of activities

$|\mathcal{A}|$: Number of precedences

$|\mathcal{L}|$: Number of learning opportunities

[1] `https://www.maplesoft.com`

λ: Learning benefit with respect to learner duration ($100 \cdot \lceil l_{i,j}/d_j \rceil$)

z^*: Optimal project makespan

$|L^*|$: Number of active learning opportunities in optimal schedule; this number corresponds to the number of insertion iterations in the algorithm

t(s): Algorithm run time in seconds

In the case that $\mathcal{G}^{\mathcal{L}}$ contains cycles, we denote this by $\circ$. If $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is cyclic, then this is indicated by $\bullet$. Note that $\circ$ implies $\circ\bullet$ but not vice versa. For 39 instances, $\mathcal{G}^{\mathcal{L}}$ is cyclic and $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ is cyclic for 156 instances.

6.2. No Learning, Organic Learning, and Optimal Learning

Table 4 shows the average benefit of learning for different instance sizes, relative number of potential learning activities ($|\mathcal{L}|\%$) and learning intensity (λ). The relative makespan reduction with respect to the optimal non-learning project makespan is denoted by $\Delta\%$. Moreover, it contains the relative number of learning opportunities that are utilized ($|L^*|\%$).

It can be seen that projects are clearly finished earlier even in the case of few opportunities and low individual learning effects. We observe that both factors, the increase of learning opportunities and the increase of learning reduction have a major impact on benefit. Figure 5 reveals a near-linear increase of learning benefit. This effect is comparable to what has been observed for projects with limited resource availability (Hill et al. 2021). However, $|\mathcal{L}|\%$ is only correlated with $\Delta\%$ and does not impact $|L^*|\%$ which consistently ranges from 37.3% to 58.3% for both 30-job and 120-job projects. In our experiments, λ turns out to be the main driver for makespan improvements. The highest individual makespan saving that occurs is 57.9% – from 57 to 24 (instance 81).

To better understand the effectiveness of our sub-optimal optimization methods, we illustrate average relative makespan reductions in Figure 6. Although organic learning is effective, only our optimal strategy unleashes the remaining 11% of the achievable overall learning benefit.

6.3. Multiple Learning Opportunities

We generated 160 instances for the more general case with multiple learning opportunities – 80 with 30 jobs and 80 with 120 jobs – as follows. For each base PSP-SL instance (see Section 6.1) with a medium learning impact ($\lambda = 50$), we created four scenarios in which we increased the number of learning opportunities by 20%, 40%, 60%, and 80%, respectively.

Table 2 Detailed properties and results for PSP-SL 30-job instances.

#	$\|\mathcal{V}\|$	$\|\mathcal{A}\|$	$\|\mathcal{L}\|$	λ	○●	z^*	$\|L^*\|$	t(s)	#	$\|\mathcal{V}\|$	$\|\mathcal{A}\|$	$\|\mathcal{L}\|$	λ	○●	z^*	$\|L^*\|$	t(s)
1	30	48	6	10		38	3	0.0	**46**	30	58	6	10	●	48	2	0.1
2	30	48	6	50		38	3	0.0	**47**	30	58	6	50	●	45	2	0.1
3	30	48	6	90		38	3	0.0	**48**	30	58	6	90	●	40	3	0.1
4	30	48	15	10	●	34	7	0.1	**49**	30	58	15	10	○●	47	5	0.3
5	30	48	15	50	●	30	7	0.2	**50**	30	58	15	50	○●	41	6	0.3
6	30	48	15	90	●	28	7	0.1	**51**	30	58	15	90	○●	37	7	0.2
7	30	48	24	10	●	33	9	0.3	**52**	30	58	24	10	○●	46	11	0.5
8	30	48	24	50	●	29	9	0.3	**53**	30	58	24	50	○●	38	11	0.6
9	30	48	24	90	●	27	11	0.3	**54**	30	58	24	90	○●	31	14	0.5
10	30	48	6	10		46	3	0.0	**55**	30	68	6	10		58	2	0.1
11	30	48	6	50		40	4	0.1	**56**	30	68	6	50		53	3	0.1
12	30	48	6	90		35	4	0.1	**57**	30	68	6	90		46	4	0.1
13	30	48	15	10	●	45	6	0.3	**58**	30	68	15	10	●	57	3	0.3
14	30	48	15	50	●	38	8	0.3	**59**	30	68	15	50	●	47	6	0.3
15	30	48	15	90	●	33	8	0.3	**60**	30	68	15	90	●	35	9	0.3
16	30	48	24	10	●	44	8	0.6	**61**	30	68	24	10	●	55	6	0.6
17	30	48	24	50	●	35	12	0.6	**62**	30	68	24	50	●	44	9	0.5
18	30	48	24	90	●	29	13	0.6	**63**	30	68	24	90	●	32	10	0.5
19	30	48	6	10		61	3	0.1	**64**	30	68	6	10		54	2	0.1
20	30	48	6	50		55	4	0.1	**65**	30	68	6	50		49	2	0.1
21	30	48	6	90		49	4	0.1	**66**	30	68	6	90		47	1	0.1
22	30	48	15	10	○●	59	6	0.3	**67**	30	68	15	10	○●	53	6	0.3
23	30	48	15	50	○●	48	8	0.3	**68**	30	68	15	50	○●	48	6	0.3
24	30	48	15	90	○●	37	8	0.3	**69**	30	68	15	90	○●	43	7	0.3
25	30	48	24	10	○●	59	10	0.5	**70**	30	68	24	10	○●	52	9	0.6
26	30	48	24	50	○●	47	11	0.5	**71**	30	68	24	50	○●	45	9	0.6
27	30	48	24	90	○●	34	13	0.5	**72**	30	68	24	90	○●	38	10	0.6
28	30	58	6	10		51	3	0.1	**73**	30	68	6	10	●	56	2	0.1
29	30	58	6	50		46	5	0.1	**74**	30	68	6	50	●	50	3	0.1
30	30	58	6	90		44	5	0.1	**75**	30	68	6	90	●	48	4	0.1
31	30	58	15	10		50	5	0.3	**76**	30	68	15	10	●	54	7	0.3
32	30	58	15	50		41	9	0.3	**77**	30	68	15	50	●	44	8	0.3
33	30	58	15	90		37	10	0.3	**78**	30	68	15	90	●	37	8	0.3
34	30	58	24	10	○●	49	10	0.5	**79**	30	68	24	10	●	53	10	0.6
35	30	58	24	50	○●	37	13	0.6	**80**	30	68	24	50	●	38	13	0.6
36	30	58	24	90	○●	28	17	0.6	**81**	30	68	24	90	●	24	15	0.6
37	30	58	6	10		61	2	0.1	**82**	30	68	6	10	●	53	3	0.1
38	30	58	6	50		54	3	0.1	**83**	30	68	6	50	●	50	3	0.1
39	30	58	6	90		45	4	0.1	**84**	30	68	6	90	●	47	3	0.1
40	30	58	15	10	●	61	7	0.3	**85**	30	68	15	10	●	53	4	0.3
41	30	58	15	50	●	54	7	0.3	**86**	30	68	15	50	●	48	5	0.3
42	30	58	15	90	●	44	10	0.3	**87**	30	68	15	90	●	43	5	0.3
43	30	58	24	10	○●	60	11	0.5	**88**	30	68	24	10	●	52	7	0.5
44	30	58	24	50	○●	46	12	0.6	**89**	30	68	24	50	●	45	9	0.6
45	30	58	24	90	○●	34	16	0.6	**90**	30	68	24	90	●	37	11	0.6

Table 3 Detailed properties and results for PSP-SL 120-job instances.

#	$\|\mathcal{V}\|$	$\|\mathcal{A}\|$	$\|\mathcal{L}\|$	λ	○●	z^*	$\|L^*\|$	t(s)	#	$\|\mathcal{V}\|$	$\|\mathcal{A}\|$	$\|\mathcal{L}\|$	λ	○●	z^*	$\|L^*\|$	t(s)
91	120	183	30	10	●	87	10	1.7	**136**	120	257	30	10	●	99	9	1.8
92	120	183	30	50	●	83	11	2.6	**137**	120	257	30	50	●	92	10	1.8
93	120	183	30	90	●	79	13	2.0	**138**	120	257	30	90	●	87	10	1.8
94	120	183	60	10	●	86	20	7.3	**139**	120	257	60	10	●	98	17	5.8
95	120	183	60	50	●	80	23	5.7	**140**	120	257	60	50	●	80	22	5.8
96	120	183	60	90	●	74	26	4.8	**141**	120	257	60	90	●	70	24	5.5
97	120	183	90	10	●	82	33	9.3	**142**	120	257	90	10	○●	95	34	10.7
98	120	183	90	50	●	67	38	9.4	**143**	120	257	90	50	○●	78	38	11.1
99	120	183	90	90	●	55	44	9.7	**144**	120	257	90	90	○●	58	46	11.7
100	120	220	30	10	●	95	15	1.9	**145**	120	257	30	10	●	84	13	1.7
101	120	220	30	50	●	80	19	2.0	**146**	120	257	30	50	●	75	13	1.7
102	120	220	30	90	●	69	19	2.0	**147**	120	257	30	90	●	71	17	1.8
103	120	220	60	10	●	92	28	6.6	**148**	120	257	60	10	●	81	28	5.3
104	120	220	60	50	●	72	32	7.1	**149**	120	257	60	50	●	71	27	6.2
105	120	220	60	90	●	61	36	5.9	**150**	120	257	60	90	●	65	30	5.1
106	120	220	90	10	●	92	41	11.8	**151**	120	257	90	10	●	77	40	10.8
107	120	220	90	50	●	70	49	11.2	**152**	120	257	90	50	●	65	42	11.5
108	120	220	90	90	●	54	52	10.8	**153**	120	257	90	90	●	57	48	10.9
109	120	220	30	10	●	100	13	2.0	**154**	120	257	30	10	●	130	14	1.6
110	120	220	30	50	●	94	14	1.7	**155**	120	257	30	50	●	119	14	1.8
111	120	220	30	90	●	88	16	1.7	**156**	120	257	30	90	●	111	14	1.9
112	120	220	60	10	●	97	24	5.5	**157**	120	257	60	10	●	126	27	6.0
113	120	220	60	50	●	87	29	5.3	**158**	120	257	60	50	●	101	30	6.0
114	120	220	60	90	●	75	32	5.3	**159**	120	257	60	90	●	84	31	6.2
115	120	220	90	10	○●	96	32	11.3	**160**	120	257	90	10	●	124	40	11.1
116	120	220	90	50	○●	82	37	11.0	**161**	120	257	90	50	●	95	43	10.7
117	120	220	90	90	○●	63	44	10.3	**162**	120	257	90	90	●	74	45	10.3
118	120	220	30	10	●	96	11	1.9	**163**	120	257	30	10	●	100	11	1.6
119	120	220	30	50	●	94	13	1.9	**164**	120	257	30	50	●	93	12	1.8
120	120	220	30	90	●	93	16	1.6	**165**	120	257	30	90	●	87	14	1.7
121	120	220	60	10	●	93	23	6.3	**166**	120	257	60	10	●	99	19	5.1
122	120	220	60	50	●	82	26	5.2	**167**	120	257	60	50	●	91	22	5.4
123	120	220	60	90	●	74	29	5.5	**168**	120	257	60	90	●	85	25	5.4
124	120	220	90	10	○●	92	36	10.7	**169**	120	257	90	10	○●	95	34	11.7
125	120	220	90	50	○●	74	42	11.3	**170**	120	257	90	50	○●	78	35	11.4
126	120	220	90	90	○●	58	47	10.7	**171**	120	257	90	90	○●	68	41	11.2
127	120	183	30	10	●	68	13	1.6	**172**	120	183	30	10	●	79	12	1.5
128	120	183	30	50	●	64	14	1.9	**173**	120	183	30	50	●	77	15	1.5
129	120	183	30	90	●	61	17	1.6	**174**	120	183	30	90	●	75	15	1.4
130	120	183	60	10	●	67	28	5.4	**175**	120	183	60	10	●	76	28	5.0
131	120	183	60	50	●	61	29	5.7	**176**	120	183	60	50	●	68	32	5.0
132	120	183	60	90	●	55	32	5.3	**177**	120	183	60	90	●	61	35	5.1
133	120	183	90	10	●	66	40	9.9	**178**	120	183	90	10	○●	73	40	12.0
134	120	183	90	50	●	53	42	10.6	**179**	120	183	90	50	○●	58	46	9.7
135	120	183	90	90	●	44	44	10.4	**180**	120	183	90	90	○●	46	51	10.5

Table 4 **Average relative makespan reduction and average learning utilization for different learning parameters and project sizes.**

$\lvert\mathcal{V}\rvert = 30$				$\lvert\mathcal{V}\rvert = 120$			
$\lvert\mathcal{L}\rvert\%$	λ	$\lvert L^*\rvert\%$	$\Delta\%$	$\lvert\mathcal{L}\rvert\%$	λ	$\lvert L^*\rvert\%$	$\Delta\%$
20	10	41.7	2.7	25	10	40.3	2.7
	50	53.3	10.9		50	45.0	9.5
	90	58.3	18.2		90	50.3	14.5
	*	51.1	10.6		*	45.2	8.9
50	10	37.3	5.4	50	10	40.3	5.0
	50	46.7	18.9		50	45.3	17.2
	90	52.7	30.5		90	50.0	26.3
	*	45.6	18.3		*	45.2	16.2
80	10	37.9	7.3	75	10	41.1	7.5
	50	45.0	25.2		50	45.8	25.2
	90	54.2	41.3		90	51.3	39.9
	*	45.7	24.6		*	46.1	24.2
*	*	47.5	17.8	*	*	45.5	16.4

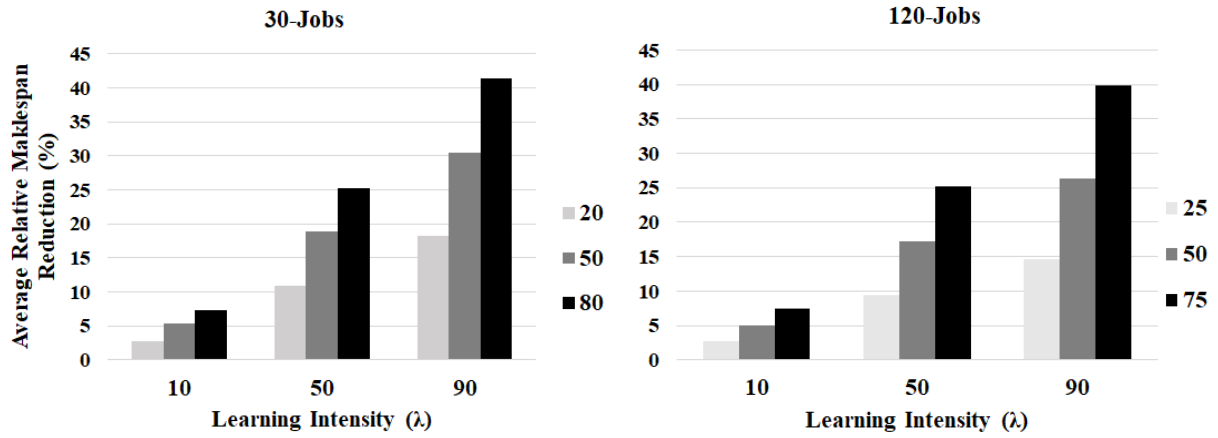

Figure 5 **The average makespan reduction (in percent) achieved through optimal learning integration for different learning intensities** *lambda* **(10%, 50%, 90%), relative number of potential learners (20/25%, 50%, 75/80%), and project sizes (30/120 jobs).**

To expand the learning network $\mathcal{G}^{\mathcal{L}}$, we randomly selected an existing learner (i.e., a job j such that $|\{(i,j) \in \mathcal{L}\}| > 0$) and a new activity to learn from. In the case that either free or impossible learning would have been induced, we re-sampled a different activity. Similarly,

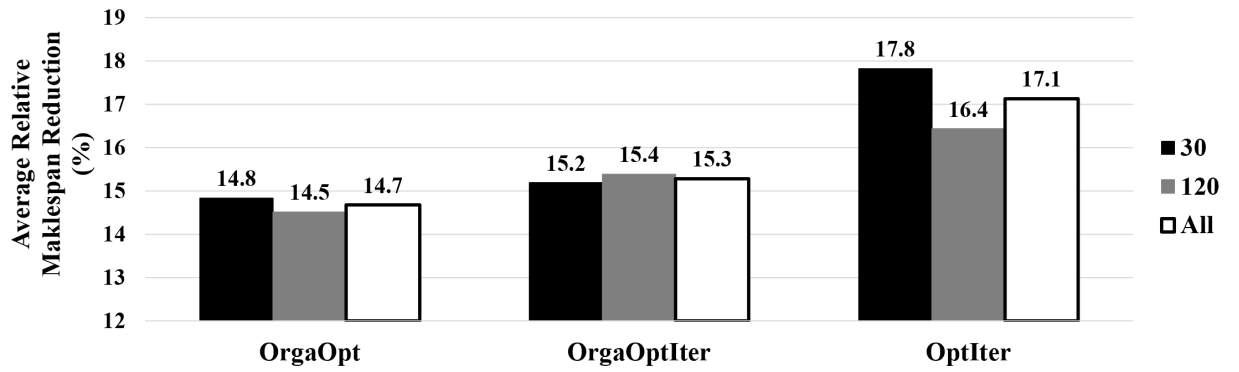

Figure 6 **The average makespan reduction when learning organically (OrgaOpt), iteratively applying organic learning (OrgaOptIter), and optimal full learning integration (OptIter) for test projects with 30 jobs (black), 120 jobs (gray), and all projects (white).**

we discarded the random candidate when our learning function $l_j(L)$ could not be extended for another teacher. We use different random seeds for the base instances, but the same seed for different scenarios to ensure that the expansion is incremental. We limited ourselves to the twenty base instances with the highest index (#) for both categories, 30 and 120 jobs.

For an activity $j \in \mathcal{V}$ and its potential learning opportunities L_j, we define the learning function to be $l_j(L') = l_j + |L'| - 1$ for $L' \subseteq L_j$. It is easy to see that this function is monotone. The detailed instance data and results obtained by Algorithm 5 are shown in Table 5.

As expected, we observe reduced acyclicity for scenarios with an increased number of learning opportunities. From 29 acyclic learning networks in the PSP-SL, the number goes down to 14 (out of 40) in the 80% scenario. Similarly, the number of acyclic combined networks $\mathcal{G}^{\mathcal{A}+\mathcal{L}}$ drops from 4 to 2. The average number of jobs that experience multi-learning in the optimal schedule is 1.6, 3.3, 4.8, and 6.3 for the augmentation scenarios (the maximum ranged from 7 to 25).

Most importantly, our experiments show that the addition of multi-learning opportunities results in a notable reduction of the optimal makespan. This effect is shown in Table 5. We observe that the average relative makespan reduction is linearly correlated with the percentage of new learning opportunities ($R = 0.996$). We note that stronger makespan reductions can be expected when using a more aggressive learning function $l_j(L)$. Our experiment choice can be considered conservative since the additional benefit of one more teacher is always one time unit. In other words, using the first teacher can be expected to be most beneficial which results in moderate attractiveness for more teachers.

Table 5 Detailed instance properties and results for PSP-ML learning opportunity scenarios, and the average relative makespan decrease (bottom line).

PSP-SL				**PSP-ML**										
#	0%			20%			40%			60%			80%		
	$\|\mathcal{L}\|$	○●	z^*	$\|\mathcal{L}\|$	○●	z^*	$\|\mathcal{L}\|$	○●	z^*	$\|\mathcal{L}\|$	○●	z^*	$\|\mathcal{L}\|$	○●	z^*
32	15		41	18		41	21	●	41	24	●	40	27	●	39
35	24	○●	37	29	○●	37	34	○●	36	39	○●	36	44	○●	34
38	6		54	8	●	54	9	●	50	10	●	49	11	●	48
41	15	●	54	18	○●	50	21	○●	50	24	○●	50	27	○●	47
44	24	○●	46	29	○●	46	34	○●	44	39	○●	44	44	○●	40
47	6	●	45	8	●	42	9	●	41	10	●	41	11	●	41
50	15	○●	41	18	○●	41	21	○●	41	24	○●	39	27	○●	39
53	24	○●	38	29	○●	37	34	○●	37	39	○●	37	44	○●	32
56	6		53	8		53	9		53	10		52	11		52
59	15	●	47	18	●	44	21	●	44	24	●	42	27	○●	39
62	24	●	44	29	●	41	34	○●	38	39	○●	37	0	○●	0
65	6		49	8		49	9		49	10		49	11		49
68	15	○●	48	18	○●	48	21	○●	47	24	○●	45	27	○●	44
71	24	○●	45	29	○●	45	34	○●	45	39	○●	45	44	○●	43
74	6	●	50	8	●	50	9	●	50	10	●	49	11	●	49
77	15	●	44	18	●	44	21	●	44	24	●	44	27	○●	43
80	24	●	38	29	○●	36	34	○●	33	39	○●	33	44	○●	33
83	6	●	50	8	●	48	9	●	48	10	●	48	11	●	46
86	15	●	48	18	○●	46	21	○●	45	24	○●	44	27	○●	42
89	24	●	45	29	●	42	34	○●	41	39	○●	39	44	○●	35
122	60	●	82	72	●	80	84	●	80	96	●	80	108	●	79
125	90	○●	74	108	○●	73	126	○●	72	144	○●	68	162	○●	67
128	30	●	64	36	●	63	42	●	63	48	●	63	54	●	62
131	60	●	61	72	●	61	84	●	60	96	●	59	108	○●	59
134	90	●	53	108	●	51	126	●	51	144	●	50	162	○●	50
137	30	●	92	36	○●	91	42	○●	91	48	○●	89	0	○●	0
140	60	●	80	72	●	80	84	○●	80	96	○●	80	108	○●	77
143	90	○●	78	108	○●	71	126	○●	71	144	○●	68	162	○●	65
146	30	●	75	36	●	75	42	●	75	48	●	75	54	●	75
149	60	●	71	72	○●	68	84	○●	67	96	○●	67	108	○●	67
152	90	●	65	108	●	65	126	○●	62	144	○●	57	162	○●	56
155	30	●	119	36	●	119	42	●	114	48	●	113	54	●	113
158	60	●	101	72	●	97	84	●	94	96	●	90	108	●	88
161	90	●	95	108	○●	89	126	○●	88	144	○●	79	162	○●	75
164	30	●	93	36	●	92	42	●	86	48	●	86	0	○●	0
167	60	●	91	72	●	88	84	●	86	96	●	86	108	●	85
170	90	○●	78	108	○●	72	126	○●	70	144	○●	70	162	○●	69
173	30	●	77	36	●	77	42	●	77	48	●	77	54	●	77
176	60	○●	68	72	○●	68	84	○●	68	96	○●	68	108	○●	67
179	90	○●	58	108	○●	57	126	○●	56	144	○●	56	162	○●	53
Δ (%)			18.0			19.9			21.3			22.6			24.8

7. Conclusion

This paper considers the impact of learning in project scheduling problems. To capture learning effects, we introduce a new class of conditional precedence relationships that allow a reduction in the duration of selected activities only if these activities are scheduled after similar activities that provide relevant experience. We study both situations where an activity learns from at most one preceding activity and where an activity can simultaneously learn from multiple activities.

The resulting project scheduling problem with learning has to determine which of these learning opportunities to activate. This is complicated by the fact the resulting learning network can introduce cycles in the overall precedence graph, and a critical challenge is therefore to determine a collection of learning opportunities that will eliminate cycles such that overall makespan is minimized. We first analyze reactive approaches that repeatedly add organic learning opportunities that might be present. The limitations of these approaches lead us to consider a more proactive approach that relies on repeatedly identifying a select subset of candidate learning opportunities. We show that the resulting algorithm yields an optimal solution in polynomial time. Using an extensive collection of well-known instances in the project scheduling literature, we conduct numerical experiments to analyze the impact of learning in project scheduling problems. Our results indicate that proactively accounting for learning opportunities can have significant benefits and lead to substantial makespan reductions.

Finally, we believe that our approach presents several opportunities for further research. It could be useful to explore mathematical programming formulations for project scheduling problem with learning that potentially lead to novel methods for the resource-constrained case. In addition, it could be interesting to explore domain-specific applications of our strategies to better understand the improvement potential in different industries. Similarly, the investigation of alternative learning curves could yield new algorithmic insights and allow the application of the presented concepts to further domains.

Acknowledgments

We thank the journal editor Alice Smith and the area editor Pascal Van Hentenryck for handling our manuscript, and the associate editor and the three reviewers for their constructive comments.

References

Ahuja RK, Magnanti TL, Orlin JB (1988) *Network flows* (Pearson).

Artigues C (2017) On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters* 45(2):154–159.

Babu A, Suresh N (1996) Project management with time, cost, and quality considerations. *European Journal of Operational Research* 88(2):320–327.

Bai D, Tang M, Zhang ZH, Santibanez-Gonzalez ED (2018) Flow shop learning effect scheduling problem with release dates. *Omega* 78:21–38.

Biskup D (2008) A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research* 188(2):315–329.

Cao H, Hall NG, Wan G, Zhao W (2024) Optimal intraproject learning. *Manufacturing & Service Operations Management* 26(2):681–700.

De Reyck B, Herroelen W (1999) The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research* 119(2):538–556.

Dodin B, Elimam A (1997) Audit scheduling with overlapping activities and sequence-dependent setup costs. *European Journal of Operational Research* 97(1):22–33.

Ebbinghaus H (1885) *Über das Gedächtnis. Untersuchungen zur experimentellen Psychologie* (Leipzig, Germany: Duncker & Humblot).

Elmaghraby SE, Kamburowski J (1992) The analysis of activity networks under generalized precedence relations (GPRs). *Management Science* 38(9):1245–1263.

Glock CH, Grosse EH, Jaber MY, Smunt TL (2019) Applications of learning curves in production and operations management: A systematic literature review. *Computers & Industrial Engineering* 131:422–441.

Hall NG (2012) Project management: Recent developments and research opportunities. *Journal of Systems Science and Systems Engineering* 21(2):129–143.

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207(1):1–14.

Hartmann S, Briskorn D (2022) An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 297(1):1–14.

Heuser P, Letmathe P, Vossen TW (2022) Skill development in the field of scheduling - a structured literature review. *Working Paper* .

Hill A, Ticktin J, Vossen TWM (2021) A computational study of constraint programming approaches for resource-constrained project scheduling with autonomous learning effects. Stuckey PJ, ed., *Integration*

of Constraint Programming, Artificial Intelligence, and Operations Research - 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5-8, 2021, Proceedings, volume 12735 of *Lecture Notes in Computer Science*, 26–44 (Springer).

Hill A, Vossen TWM (2024) Project Scheduling with Autonomous Learning Opportunities Experiments. URL `http://dx.doi.org/10.1287/ijoc.2023.0107.cd`, available for download at https://github.com/INFORMSJoC/2023.0107.

Hochbaum DS (2016) A polynomial time repeated cuts algorithm for the time cost tradeoff problem: The linear and convex crashing cost deadline problem. *Computers & Industrial Engineering* 95:64–71.

Johnson TJR (1967) *An algorithm for the resource constrained project scheduling problem.* Ph.D. thesis, Massachusetts Institute of Technology, USA.

Kelley Jr JE (1961) Critical-path planning and scheduling: Mathematical basis. *Operations Research* 9(3):296–320.

Kolisch R, Sprecher A (1997) PSPLIB-A project scheduling problem library: OR software-ORSEP operations research software exchange program. *European Journal of Operational Research* 96(1):205–216.

Korytkowski P, Malachowski B (2019) Competence-based estimation of activity duration in IT projects. *European Journal of Operational Research* 275(2):708–720.

Laborie P, Rogerie J, Shaw P, Vilím P (2018) IBM ILOG CP Optimizer for scheduling. *Constraints - An International Journal* 23(2):210–250.

Lee WC, Wu CC, Hsu PH (2010) A single-machine learning effect scheduling problem with release times. *Omega* 38(1-2):3–11.

Möhring RH, Schulz AS, Stork F, Uetz M (1999) Resource-constrained project scheduling: Computing lower bounds by solving minimum cut problems. *European Symposium on Algorithms*, 139–150 (Springer).

Möhring RH, Skutella M, Stork F (2004) Scheduling with and/or precedence constraints. *SIAM Journal on Computing* 33(2):393–415.

Peteghem VV, Vanhoucke M (2015) Influence of learning in resource-constrained project scheduling. *Computers & Industrial Engineering* 87:569–579.

Pritsker AAB, Waiters LJ, Wolfe PM (1969) Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science* 16(1):93–108.

Qian J, Steiner G (2013) Fast algorithms for scheduling with learning effects and time-dependent processing times on a single machine. *European Journal of Operational Research* 225(3):547–551.

Reyck BD, Demeulemeester E, Herroelen W (1999) Algorithms for scheduling projects with generalized precedence relations. *Project Scheduling*, 77–105 (Springer).

Schwindt C, Zimmermann J, et al. (2015) *Handbook on Project Management and Scheduling* (Springer).

Van Peteghem V, Vanhoucke M (2015) Influence of learning in resource-constrained project scheduling. *Computers & Industrial Engineering* 87:569–579.

Vanhoucke M (2008) Setup times and fast tracking in resource-constrained project scheduling. *Computers & Industrial Engineering* 54(4):1062–1070.

Vanhoucke M, Debels D (2008) The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects. *Computers & Industrial Engineering* 54(1):140–154.

Wright TP (1936) Factors affecting the cost of airplanes. *Journal of the Aeronautical Sciences* 3(4):122–128.

Yelle LE (1979) The learning curve: Historical review and comprehensive survey. *Decision Sciences* 10(2):302–328.