Wohlert, Lena Sophie; Zimmermann, Jürgen



# Resource overload problems with tardiness penalty: structural properties and solution approaches

Annals of Operations Research

**Provided in Cooperation with:**
Springer Nature

*Suggested Citation:* Wohlert, Lena Sophie; Zimmermann, Jürgen (2024) : Resource overload problems with tardiness penalty: structural properties and solution approaches, Annals of Operations Research, ISSN 1572-9338, Springer US, New York, NY, Vol. 338, Iss. 1, pp. 151-172, https://doi.org/10.1007/s10479-023-05789-2

This Version is available at:
https://hdl.handle.net/10419/315291

Annals of Operations Research (2024) 338:151–172
https://doi.org/10.1007/s10479-023-05789-2

ORIGINAL RESEARCH

# Resource overload problems with tardiness penalty: structural properties and solution approaches

**Lena Sophie Wohlert**[1] · **Jürgen Zimmermann**[1]

Received: 7 December 2022 / Accepted: 11 December 2023 / Published online: 16 January 2024


**Abstract**
In this paper, we consider a resource overload problem and add a tardiness penalty to the objective function when a prescribed project makespan is exceeded, which enables a trade-off between a balanced resource utilization and a project delay. For the tardiness penalty, we distinguish between a constant and variable delay cost variant. Based on the structural properties of the resource overload problem, we show that the search space of the resource overload problem with tardiness penalty can also be reduced utilizing quasistable schedules. In addition, we discuss the application of these findings to further problems, which include objectives composed of a locally concave and a concave function or a reward structure for an early project completion instead of a tardiness penalty. As solution approaches, we present mixed-integer linear model formulations as well as a novel genetic algorithm with a decoding procedure, which exploits the devised structural properties. The performance of the genetic algorithm is improved by implementing learning methods and utilizing lower bounds. Finally, we present results from experiments on small to medium sized problem instances.

**Keywords** Project scheduling · Resource overload · Tardiness penalty · Quasistable schedules · Genetic algorithm

## 1 Introduction

Instances of the resource overload problems often arise when costly fluctuations in resource utilization are to be minimized and a maximum project duration is given. However, as the scheduling of projects is an inherent multi-objective optimization problem, the exceeding of the prescribed project makespan can be allowed at additional cost. This enables a trade-off between a more balanced resource utilization and a project completion delay to achieve a more cost and resource efficient outcome. These additional cost may represent opportunity

✉ Lena Sophie Wohlert
lena.sophie.wohlert@tu-clausthal.de

Jürgen Zimmermann
juergen.zimmermann@tu-clausthal.de

[1] Institute of Management and Economics, Clausthal University of Technology, Julius-Albert-Str. 2, 38678 Clausthal-Zellerfeld, Germany

cost for further orders, an earlier market entry, or alternatively tardiness penalties for missed deadlines (Schnabel et al., 2018).

For the resource overload problem and other resource levelling problems several mixed-integer linear models are presented in Rieck et al. (2012) and Rieck and Zimmermann (2015). To solve the resource overload problem heuristically, Ballestin et al. (2007) propose a population-based greedy algorithm that constructs quasistable schedules, among which there is at least one optimum. Schnabel et al. (2018) consider a resource-constrained project scheduling problem where the objective includes revenues, which decrease with an increasing makespan, as well as resource overload cost. They observe that for this problem, unlike the resource overload problem, there is not always an optimum among the quasistable schedules, which would allow for a search space reduction. To solve the proposed problem, they present a genetic algorithm with different representations based on an activity list and either a maximum allowable overload per resource or per activity. Atan and Eren (2018) discuss the impact of relaxing a prescribed project makespan at no additional cost for different resource levelling problems to identify the minimum project duration for the best levelled schedule. As an heuristic, the authors use the population-based approach of Ballestin et al. (2007) but do not restore the quasistableness of the schedules. In Kim et al. (2005), the project duration is extended stepwise in increments of one time unit to determine different schedules for a project manager to decide from. Hegazy (1999) present objectives that consider the sum of the project duration and a resource levelling metric, which can be weighted. Koulinas and Anagnostopoulos (2012) formalize the approach of Hegazy (1999) further by introducing a trade-off coefficient to weight the cost factors in the objective function. Ponz-Tienda et al. (2013) present an adaptive genetic algorithm for a resource levelling problem in which exceeding the prescribed project makespan is allowed with a tardiness penalty that increases linearly with the project extension. In Hartmann and Briskorn (2022), an overview of further incorporations of makespan-dependent cost and respectively the minimization of the project makespan into various project scheduling problems is given. For instance, Shadrokh and Kianfar (2007) and Gerhards and Stürck (2018) consider a resource investment problem with tardiness penalty. An overview of the discussed literature related to resource levelling is given in Table 1, focusing on the utilized objective function and solution approach.

In this paper, we discuss the resource overload problem with different variants of tardiness penalties. The latter applies if a prescribed project makespan is exceeded, whereby a distinction is made between constant and variable cost for each time unit of delay. We close the research gap outlined in Schnabel et al. (2018) by proofing that the problem can be divided into subproblems for which there is at least one optimum among the quasistable schedules. As a result, the search space can be reduced, and in most cases only a considerably smaller subset of schedules needs to be investigated in order to obtain an optimal solution. The findings obtained are of particular importance since they also apply to other bi-objective problems whose objective function is a weighted sum of a locally concave function and a concave function. Besides mixed-integer linear model formulations, we present a novel genetic algorithm with a decoding procedure, which constructs only solutions within the derived set of quasistable schedules of different project durations. In addition, the solution representation indicates a start time selection rule for each activity and two learning methods are incorporated in the algorithm. We contribute a parametrization of the resource overload problem with tardiness penalty and test the two solution approaches on small to medium sized problem instances.

This paper is organized as follows: Sect. 2 is devoted to the problem description of the resource overload problem with tardiness penalty. Some structural properties of the basic resource overload problem are presented in Sect. 3 and are adapted to consider tardiness

**Table 1** Overview of relevant literature

| First author | Year | Objective | | | Solution approaches | | |
|---|---|---|---|---|---|---|---|
| | | RL | RL + TP | RV – ROP | MIP | GA | POP w. QS |
| Hegazy | 1999 | | ✓ | | | | |
| Kim | 2005 | ✓* | | | | | |
| Ballestin | 2007 | ✓ | | | | | ✓ |
| Koulinas | 2012 | | ✓ | | | | |
| Rieck | 2012 | ✓ | | | ✓ | | |
| Ponz-Tienda | 2013 | | ✓ | | | ✓ | |
| Rieck | 2015 | ✓ | | | ✓ | | |
| Atan | 2018 | ✓* | | | ✓ | | |
| Schnabel | 2018 | | | ✓ | ✓ | ✓ | |

RL: Resource levelling (including resource overload), TP: Tardiness penalty, RV: Revenue
MIP: Mixed-integer linear program, GA: Genetic algorithm, POP w. QS: Population based approach with quasistable schedules
* different project makespans are investigated

penalty, too. The application of these findings to other (bi-objective) problems is discussed in Subsection 3.4. In Sect. 4, mathematical model formulations are presented. Section 5 introduces our novel genetic algorithm. We test the solution approaches in experiments, which are described in Sect. 6 and the paper is concluded in Sect. 7.

## 2 Problem description

We assume that a project consists of a set of activities $V$, where each activity $i \in V$ is assigned a processing time $p_i \geq 0$ that cannot be interrupted. Among the activities, there are $n$ real activities and two fictitious activities 0 and $n+1$ with $p_0 = p_{n+1} = 0$ corresponding to the project start and completion. The start time of activity $i$ is given by $S_i \geq 0$, where the project start is assumed to be scheduled at time zero ($S_0 = 0$). Moreover, the completion time of an activity $i$ is defined as $C_i := S_i + p_i$. Between the start times of two activities $i$ and $j$, general temporal constraints $S_j - S_i \geq \delta_{ij}$ are assumed with $\delta_{ij} \in \mathbb{R}$. A minimum time lag between the start of activity $i$ and $j$ applies if $\delta_{ij} \geq 0$, whereas $\delta_{ij} < 0$ denotes a maximum time lag. Among the maximum time lags, the maximum project duration $\overline{d}$ (specifying the underlying planning horizon) is described by a temporal constraint from project completion to project start $S_0 - S_{n+1} \geq -\overline{d}$. For each activity $i$ an earliest start time $ES_i$ and latest start time $LS_i$ can be computed by means of some longest path algorithm (Ahuja et al., 1993), from which the total float $TF_i := LS_i - ES_i$ of each activity $i$ can be derived. A schedule $S = (S_0, S_1, \ldots, S_{n+1})$ contains a sequence of start times, which are ordered according to the numbering of the activities. To visualize the project, we use an activity-on-node network $N = (V, E, \delta)$, where the processing time $p_i$ is given above each node $i \in V$ and an arc $\langle i, j \rangle \in E$ depicts a minimum or maximum time lag $\delta_{ij}$.

A schedule that satisfies all the temporal constraints is termed time-feasible and the set of those schedules is denoted as $\mathcal{S}_T$. The set of renewable resources, which are utilized to carry out the activities, is represented by $\mathcal{R}$. Each activity $i$ is assigned a resource utilization $r_{ik} \geq 0$ for each resource $k \in \mathcal{R}$, which are listed below the respective activity in the project network $N$. The resource function $r_k(S, \cdot)$ of a schedule $S$ and a resource $k$ is a stepwise function,

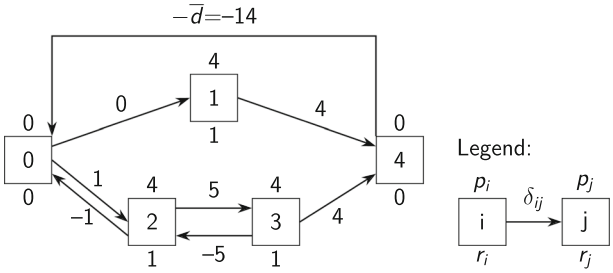


**Fig. 1** Activity-on-node project network

which computes the sum of the resource utilization of every active activity at point in time $t \in \mathbb{R}_{\geq 0}$. An activity $i$ is active if the current time $t$ is greater or equal to the start time $S_i$ and less than $C_i$. A resource profile visualizes the resource function $r_k(S, \cdot)$ for a given resource $k \in \mathcal{R}$ and schedule $S$ in dependence of $t$. The resources are assumed to be uncapacitated. However, a resource threshold $Y_k$ is defined for each resource $k \in \mathcal{R}$ from which positive deviations in the resource utilization $r_k(S, t)$ at any point in time $t$ are penalized.

***Example 1*** Figure 1 depicts an activity-on-node network which contains three real activities and one renewable resource. The earliest project completion is 10 and the maximum project duration $\overline{d}$ is limited to 14. Activity 2 has only one feasible start time $S_2 = 1$ and activity 3 has to be scheduled at $S_3 = 6$. Only activities 1 and 4 have a total float greater than zero.

The objective of the basic resource overload problem coincides with the cumulative cost of positive deviations from the resource thresholds $Y_k$. The positive deviations for each resource can be weighted differently by cost factors $c_k \in \mathbb{R}_{>0}$.

$$f_{ROP}(S) = \sum_{k \in R} c_k \int_{t \in [0, \overline{d}]} (r_k(S, t) - Y_k)^+ dt$$

The resource overload problem is to determine an optimal schedule that minimizes the objective function $f_{ROP}$ while satisfying the temporal constraints. It can be formulated as follows:

$$\begin{array}{lll} \text{Minimize} & f_{ROP}(S) & \\ \text{subject to} & S_j - S_i \geq \delta_{ij} & \langle i, j \rangle \in E \\ & S_0 = 0 & \\ & S_i \geq 0 & i \in V \end{array}$$

As there are no other constraints such as resource capacities, the set of time-feasible schedules equals the set of feasible schedules i.e. $\mathcal{S} = \mathcal{S}_T$.

For the resource overload problem with tardiness penalty, we assume that besides a maximum project duration $\overline{d}$ a prescribed project makespan $\overline{T} \in [ES_{n+1}, \overline{d}]$ is given. $\overline{T}$ can be exceeded at additional cost $f_{TP}$, which is added to the objective function (Ponz-Tienda et al., 2013; Shadrokh & Kianfar, 2007).

$$f(S) = f_{ROP}(S) + f_{TP}(S)$$

Here, we assume that the prescribed project makespan $\overline{T}$ and the maximum project duration $\overline{d}$ are positive integers. Additionally, we suppose that an increase in tardiness penalty only occurs with each whole unit of time and not with each arbitrarily small delay. First, the delay

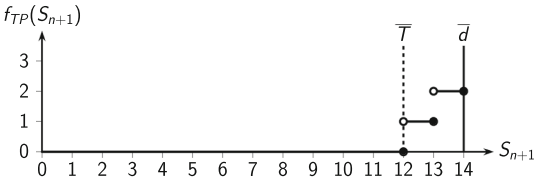


**Fig. 2** Tardiness penalty function with a constant delay cost factor $\rho = 1$

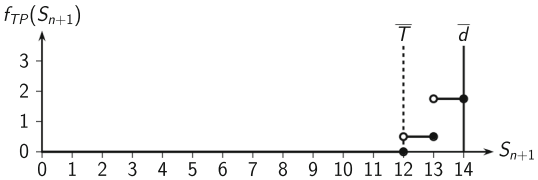


**Fig. 3** Tardiness penalty function with a variable delay cost factor $\rho_{13} = 0.5$ and $\rho_{14} = 1.25$

cost factor $\rho \in \mathbb{R}_{\geq 0}$ is assumed to be constant, with the value of $f_{TP}$ increasing linearly with an additional time unit to be penalized. The resulting tardiness penalty function $f_{TP}$ is piecewise defined depending on the project completion $S_{n+1}$ and has jump points at every integer time point greater or equal to $\overline{T}$. Moreover, $f_{TP}$ is lower semi-continuous on all schedules.

$$f_{TP}(S) = \begin{cases} 0, & S_{n+1} \in \left[ ES_{n+1}, \overline{T} \right]; \\ \rho \cdot (a+1), & S_{n+1} \in \left( \overline{T} + a, \overline{T} + a + 1 \right], a \in \left\{ 0, \ldots, \overline{d} - \overline{T} - 1 \right\}. \end{cases}$$

In typical real-world examples, different additional cost $\rho_t \geq 0, t \in \{\overline{T}+1, \ldots, \overline{d}\}$ may apply for every additional time unit of delay. The resulting function $f_{TP}$ is also lower semi-continuous and monotonically increasing.

$$f_{TP}(S) = \begin{cases} 0, & S_{n+1} \in \left[ ES_{n+1}; \overline{T} \right]; \\ \sum\limits_{t=\overline{T}+1}^{\lceil S_{n+1} \rceil} \rho_t, & S_{n+1} \in \left( \overline{T} + a; \overline{T} + a + 1 \right], a \in \left\{ 0, \ldots, \overline{d} - \overline{T} - 1 \right\}. \end{cases}$$

***Example 2*** In the following, an example is given for each tardiness penalty function variant. As in the project network in Fig. 1, a maximum project duration of 14 is assumed. Let $\overline{T} = 12$ be the prescribed project makespan. In Fig. 2, a tardiness penalty function with a constant delay cost factor $\rho = 1$ is shown. Figure 3 depicts a tardiness penalty cost function with variable, increasing, delay cost factors.

***Remark 1*** The formulation of $f_{TP}$ can be further generalized for constant and variable delay cost. Instead of no cost ($f_{TP} = 0$) for $S_{n+1} \in \left[ ES_{n+1}; \overline{T} \right]$, any constant $\rho_0 \in \mathbb{R}$ can be utilized. Obviously, for two schedules $S_1$ and $S_2$ with $f_{TP}(S_1) < f_{TP}(S_2)$, the same holds even if the constant $\rho_0$ is considered. The generalization can be used to represent a reward structure for an early project completion which may only decrease with an increase of the project duration like in Schnabel et al. (2018). However, such a function must be multiplied by $-1$ to satisfy the minimization objective.

As shown in Neumann et al. (2003), the basic resource overload problem is already $\mathcal{NP}$-hard in the strong sense. Thus, the same applies to the resource overload problem with tardiness penalty. However, finding a feasible solution is easy, because the $ES$-schedule is always feasible.

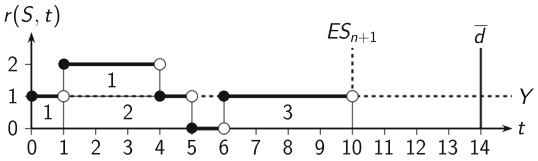


**Fig. 4** Resource profile of $ES = (0, 0, 1, 6, 10)$

## 3 Structural properties

In this section, we discuss order-based structural properties and properties of the resource overload objective function. These can be utilized to reduce the search space of the basic resource overload problem, and then similarly for variants with tardiness penalty. As described in Neumann et al. (2003), a strict order relation is implicated if an activity $j$ starts after an activity $i$ is completed ($S_j \geq S_i + p_i$). A schedule induced order $O(S)$ is introduced to refer to all strict order relations between pairs of real activities $(i, j) \in \{1, \ldots, n\} \times \{1, \ldots, n\}$ that are met by the schedule $S$. Let

$$\mathcal{S}_T(O(S)) := \{S \in \mathcal{S}_T \mid S_j \geq S_i + p_i \text{ for all } (i, j) \in O(S)\}$$

denote the set of all time-feasible schedules that satisfy the strict order given by $O(S)$. A set of schedules that induces an identical strict order as schedule $S$ is termed equal order set $\mathcal{S}_T^{=}(O(S))$. If the start time $S_i$ of at least one activity $i$ is shifted forward or backward from a time feasible schedule $S$, resulting in a non-identical and time-feasible schedule $S'$, this movement is referred to as a shift. An order preserving shift describes a shift from a schedule $S$ to another feasible schedule $S'$ such that the order of the initial schedule is preserved ($O(S') \supseteq O(S)$). Two shifts that transform a schedule $S$ into a schedule $S'$ and a schedule $S''$ are called a pair of opposite shifts if $S'' - S = \lambda(S' - S)$ with $\lambda < 0$ holds. A feasible schedule $S$ is said to be quasistable if there is no pair of opposite order preserving shifts.

### 3.1 Resource overload problem

The function $f_{ROP}$ is continuous and thus also lower semi-continuous. Moreover, the function is concave on each equal order set, denoted as locally concave. Neumann et al. (2003) proof that there is always a quasistable schedule among the optima for functions $f$ that are locally concave. A quasistable schedule $S$ has useful structural properties: For the start time of each activity $i \in V$, there is always an activity $j \in V$ for which either a precedence relationship $S_j = S_i + p_i$ or $S_j = S_i - p_j$ or a temporal constraint $S_j = S_i + \delta_{ij}$ or $S_j = S_i - \delta_{ji}$ is binding (Neumann et al., 2003). If $\delta_{ij} \in \mathbb{Z}$ (including $\overline{d}$), $p_i \in \mathbb{N}_0$ and $\mathcal{S}_T \neq \emptyset$, quasistable schedules contain only integer start times. Since there is always an optimum among quasistable schedules, there is also always an integer optimum solution. Therefore, the time horizon can be discretized for the resource overload problem without loss of solution quality. In the following, we assume that the temporal input fulfills the above mentioned conditions. In addition, if only one renewable resource is used in an example, the corresponding indices are omitted.

***Example 3*** We consider again the project depicted in Fig. 1. The resource profile of the $ES$-schedule is displayed in Fig. 4. The $ES$-schedule induces the order $O(ES) = \{(1, 3), (2, 3)\}$. In Fig. 5, $f_{ROP}$ is depicted in dependence of the start time $S_1$ of activity 1 where the cost per resource overload unit is assumed to be $c = 1$ and the resource threshold is set to be $Y = 1$. It is obvious that the minimum of $f_{ROP}(S_1)$ is at $S_1 = 10$. In the same figure, the different

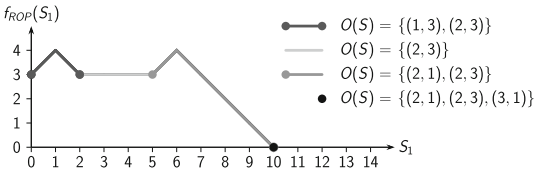


**Fig. 5** Resource overload function, schedule induced orders, equal order sets and quasistable schedules

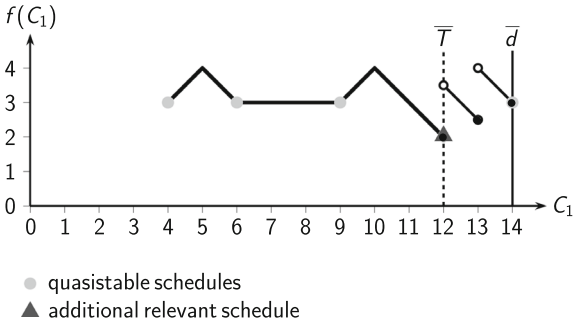


**Fig. 6** Resource overload function with a constant delay cost factor

induced orders are highlighted in different shades depending on the start time of activity 1. Since identical schedule induced orders belong to the same shade of gray, their respective equal order sets are the union of all schedules marked with the same shade. It can be seen that the resource overload function is concave on equal order sets. The quasistable schedules are marked with dots, which include the minimum of the function $f_{ROP}$.

### 3.2 Constant delay cost factor

In this subsection, we discuss structural properties of the resource overload problem with a constant delay cost factor.

***Example 4*** We consider again the project depicted in Fig. 1. For $f_{TP}$, a constant delay cost factor $\rho = c = 1$ is assumed for every additional time unit of delay when the prescribed project makespan of $\overline{T} = 12$ is exceeded (see Fig. 2). In contrast to the resource overload function in Fig. 5, $f_{ROP}$ can also be visualized as a function of the completion $C_1$ of activity 1. This is useful here because the completion of activity 1 and the project completion $S_{n+1}$ coincide if activity 1 completes at time 6 or later. Moreover, activity 1 is the only real activity that can be shifted. As a result, $f_{TP}$ and thus $f$ can also be represented as a function of $C_1$ in this specific example. In Fig. 6, the course of function $f$ is depicted. The quasistable schedules are marked in light gray. It can be seen that the minimum of the combined objective function $f$ is reached when activity 1 ends at 12, which is the prescribed project makespan $\overline{T}$. However, this schedule is not a quasistable schedule.

In the following, we provide an extension of the set of quasistable schedules that contains at least one optimum for resource overload problems with a constant delay cost factor. This set is obtained by dividing the problem into two subproblems based on different minimum and maximum project durations and determining the set of quasistable schedules for both subproblems.

**Theorem 1** *Given a resource overload problem with a constant delay cost factor $P$, two subproblems $P'$ and $P''$ can be constructed. The set of feasible schedules of subproblem $P'$*

*is further limited by an adjusted maximum project duration of* $\overline{T}$*. The set of feasible schedules of the second subproblem* $P''$ *is further limited by a minimum project duration of* $\overline{T}$*. Among the quasistable schedules of the two subproblems* $P'$ *and* $P''$*, there is at least one optimal solution of* $P$*.*

***Proof*** A resource overload problem with a constant delay cost factor can be divided into two subproblems. For the first subproblem, a maximum project duration of $\overline{T}$ applies. This subproblem has a set of feasible schedules $\mathcal{S}_{T,\leq\overline{T}}$, all of which have no tardiness penalty. Therefore, this subproblem is a basic resource overload problem and among the set of its quasistable schedules, there is at least one of its optima. The second subproblem has a minimum project duration of $\overline{T}$ and a maximum project duration of $\overline{d}$. The connected set of time-feasible schedules $\mathcal{S}_{T,\geq\overline{T}}$ contains schedules $S \in \mathcal{S}_T$ with a project duration of $\overline{T} \leq S_{n+1} \leq \overline{d}$. The equal order sets of this subproblem are denoted as

$$\mathcal{S}_T^{=}(O(S))_{\geq\overline{T}} := \mathcal{S}_T^{=}(O(S)) \cap \left\{S' \in \mathcal{S}_T : S'_{n+1} \geq \overline{T}\right\}.$$

On the equal order sets $\mathcal{S}_T^{=}(O(S))_{\geq\overline{T}}$, the resource overload function $f_{ROP}$ is still concave. However, for this subproblem the combined objective function $f$ is not equal to $f_{ROP}$ because the values of the tardiness penalty function $f_{TP}$ can be greater than zero. To nevertheless determine schedules that minimize $f$ in this subproblem, the auxiliary function $f_h$ is defined for $S \in \mathcal{S}_T$ with $S_{n+1} \geq \overline{T}$.

$$f_h(S) = \rho \cdot \left(S_{n+1} - \overline{T}\right)$$

The function $f_{TP}$ is the ceiling function of $f_h$ within the discussed subproblem. Therefore, $f_h$ underestimates the value of $f_{TP}$ at every point in time $t \in [\overline{T}, \overline{d}]$ by a value within the interval $[0, 1)$. Moreover, functions $f_h$ and $f_{TP}$ have the same objective function values if the project makespan $S_{n+1}$ is an integer and thus $f_{ROP} + f_{TP} = f_{ROP} + f_h$ for all schedules with $S_{n+1} \in \mathbb{N}$. The auxiliary function $f_h$ is concave and therefore also locally concave. The sum of locally concave functions, here $f_{ROP}$ and $f_h$, is again locally concave. Since $f_{ROP} + f_h$ underestimates $f$ and they have the same value for each schedule with only integer start times, among the quasistable schedules that satisfy $\overline{T} \leq S_{n+1} \leq \overline{d}$ is at least one optimum of this subproblem.

***Remark 2*** Let $I$ be an instance of the resource overload problem with a constant delay cost factor. The project network $N$ of $I$ can be modified to represent the temporal constraints of either of the two subproblems described. By adding an arc $\langle n+1, 0\rangle$ with the weight $\delta_{n+1,0} = -\overline{T}$, the project network $N$ is adapted to the first subproblem. For the second subproblem an arc $\langle 0, n+1\rangle$ with a weight of $\delta_{0,n+1} = \overline{T}$ is added, which indicates a minimum time lag and therefore a minimum project duration.

### 3.3 Variable delay cost factor

We now assume that for every additional time unit of delay different additional tardiness penalty cost $\rho_t \geq 0$, with $t \in \left[\overline{T}+1, \ldots, \overline{d}\right]$ apply, which results in a monotonically increasing penalty function. This function cannot be necessarily underestimated by a linear function. Moreover, it can be seen in Fig. 7 that the minimum is not among the schedules described in Theorem 1. The problem is therefore divided into further subproblems.

**Theorem 2** *Given a resource overload problem with a variable delay cost factor, the following subproblems can be constructed. The subproblem* $P'$ *has a set of feasible schedules that is*

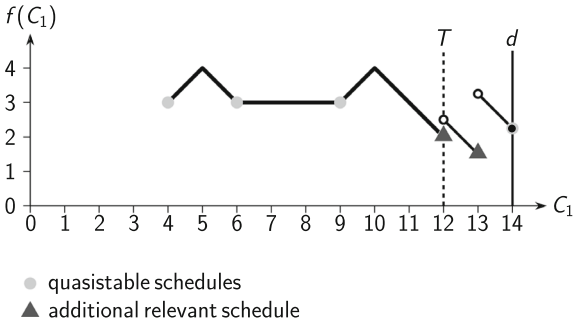


**Fig. 7** Resource overload function with a variable delay cost factor

*further limited by an adjusted maximum project duration of* $\overline{T}$. *For all* $a \in \{0, \ldots, \overline{d} - \overline{T} - 1\}$ *an additional subproblem* $P^a$ *is defined. The set of feasible schedules of each* $P^a$ *is further limited by a minimum project duration of* $\overline{T}+a$ *and by a maximum project duration of* $\overline{T}+a+1$. *Among the quasistable schedules of the subproblems* $P'$ *and* $P^a$ *for all* $a \in \{0, \ldots, \overline{d}-\overline{T}-1\}$, *there is at least one optimal solution of* $P$.

***Proof*** It can be argued analogously to the previous proof. However, the resource overload problem with variable delay cost factors is divided into $\overline{d} - \overline{T} + 1$ subproblems. The first subproblem has a maximum project duration of $\overline{T}$. In this subproblem no tardiness penalty occurs and therefore it is a basic resource overload problem. The additional subproblems are distinguished by their minimum and maximum project duration, which are computed in dependence of $a \in \{0, \ldots, \overline{d} - \overline{T} - 1\}$. The minimum project duration of each of these subproblems is equal to $\overline{T} + a$ and the maximum project duration is equal to $\overline{T} + a + 1$. For each of these subproblems, a set of feasible schedules $\mathcal{S}_{T,a}$ and equal order sets $\mathcal{S}^{=}_{\overline{T},a}(O(S))$ can be defined as

$$\mathcal{S}_{T,a} := \mathcal{S}_T \cap \left\{S \in \mathcal{S}_T : \overline{T} + a \leq S_{n+1} \leq \overline{T} + a + 1\right\}$$

and

$$\mathcal{S}^{=}_{\overline{T},a}(O(S)) := \mathcal{S}^{=}_{\overline{T}}(O(S)) \cap \left\{S' \in \mathcal{S}_T : \overline{T} + a \leq S'_{n+1} \leq \overline{T} + a + 1\right\}.$$

Again an auxiliary function $f_h$ can be defined in dependence of $S_{n+1}$ for $S \in \mathcal{S}_T$.

$$f_h(S) = \sum_{t=\overline{T}+1}^{\lceil S_{n+1} \rceil - 1} \rho_t + (S_{n+1} - \lceil S_{n+1} \rceil - 1) \cdot \rho_{\lceil S_{n+1} \rceil - 1}$$

Because $f_{TP}(S)$ equals the ceil of $f_h(S)$ for every schedule $S \in \mathcal{S}_{T,a}$, the latter function underestimates the value of the former $f_{TP}$. Moreover, the functions $f_h$ and $f_{TP}$ have the same objective function values for each integer project duration. The auxiliary function $f_h$ is concave on every subproblem and therefore also concave on every equal order set $\mathcal{S}^{=}_{\overline{T},a}(O(S))$ but not necessarily concave beyond the described subsets. The sum of $f_h$ and $f_{ROP}$ is again concave on $\mathcal{S}^{=}_{\overline{T},a}(O(S))$. Because the sum of $f_h$ and $f_{ROP}$ underestimates $f$ in the respective subproblem and has the same value for each schedule with only integer start times, among the set of quasistable schedules of each described subproblem is at least one of its optima.

### 3.4 Comprehensive application of findings

The structural properties devised can also be used to reduce the search space of other project scheduling problems including resource levelling problem variants with resource constraints or makespan-dependent resource thresholds. Generally, Theorem 1 applies to problems whose objective is a sum of a locally concave function like the resource overload function and a concave function like a linear increasing cost function. Instead of the concave function, lower semi-continuous functions that can be underestimated in the way described can be utilized. As a result, the basic resource levelling problem, which considers total squared utilization cost, and whose objective function is also locally concave could be combined with the described tardiness penalty functions into a single-objective and the search space of the problem can be reduced. The same applies for the total adjustment cost objective, where the cost result from increases and decreases in resource utilization. Structural properties of this metric are discussed in Kreter et al. (2014). The release and re-hire objective adapts the total adjustment cost objective by subtracting the maximum resource utilization for each resource (Atan & Eren, 2018). Since the maximum resource utilization for each resource is the same for all schedules within the same equal order, it can be assumed to be a constant for this set. Therefore, the release and re-hire objective is still locally concave.

As discussed, Schnabel et al. (2018) consider an objective function consisting of makespan-dependent revenue reduced by overload cost, which can be transformed into the objective function studied in this paper as described in Remark 1. However, they also apply resource constraints. In this case, not all time-feasible schedules are also resource feasible. However, all schedules within the same equal order set have the same maximum resource utilization for each resource. If one of these maximum resource utilizations violates a resource constraint, then all schedules in the equal order set are not feasible. Furthermore, if a schedule within an equal order set satisfies the resource constraints, all schedules in the equal order sets are feasible. Therefore, the resource overload problem with tardiness penalty and resource constraints can again be divided into the delay cost variant-dependent subproblems. For each subproblem, the objective function $f$ is still concave on each resource-feasible equal order set and the extension of the quasistable schedules contains at least one optimum.

Another problem is the resource overload problem with makespan-dependent resource threshold, which was proposed by Atan and Eren (2018). In this problem variant, the thresholds for evaluating resource deviations decreases with an increasing project makespan. Since resources are usually acquired or employed in whole units, it does not appear reasonable for the resource thresholds to be exactly the average of total resource utilization and the project makespan. If the average resource utilization is rounded to the largest integer less or equal, the resulting objective function $f$ is lower semi-continuous. To ensure that jump points only occur at integers time points, we apply the ceil function to the project makespan $S_{n+1}$.

$$f(S) = \sum_{k \in R} c_k \int_{t \in [0,\overline{d}]} \left( r_k(S,t) - \underbrace{\left\lfloor \frac{\sum_{i \in V} r_{ik} \cdot p_i}{\lceil S_{n+1} \rceil} \right\rfloor}_{Y_k(S)} \right)^+ dt$$

This problem can be divided into subproblems with the same resource thresholds $Y_k(S)$ for all $k \in \mathcal{R}$. Each of the subproblems resembles a resource overload problem with fixed resource thresholds. In contrast to the aforementioned problems, the set of feasible schedules for every subproblem is not necessarily closed. However, if this is the case the limit points of the set of feasible schedules are the boundary of another subproblem, where at least one

resource threshold is greater than in the considered subproblem. Because $f_{ROP}$ is lower semi-continuous, the schedules on the boundary of the other subproblem have an equal or lower objective value than every schedule in the first considered problem. Thus, the search space of this problem can also be restricted to quasistable schedules with different project durations.

## 4 Mathematical model formulation

Like most model formulations for project scheduling problems, we utilize a time-discrete formulation, which was introduced by Pritsker et al. (1969). Binary variables $x_{it}$ are defined for every activity $i \in V$ and for every feasible start time $t \in W_i = \{ES_i, \ldots, LS_i\}$, where

$$x_{it} = \begin{cases} 1, & \text{if activity } i \text{ starts at point in time } t, \\ 0, & \text{otherwise.} \end{cases}$$

To determine the total resource overload, non-negative overload variables $o_{kt}$ for every $k \in \mathcal{R}$ and $t \in T := \{0, \ldots, \overline{d} - 1\}$ are utilized (Rieck & Zimmermann, 2015). The formulation for the resource overload problem with a constant delay cost factor is given in (1)–(9), where the auxiliary variable $\Delta\overline{T}$ is utilized to represent the delay of the prescribed project makespan like in Ponz-Tienda et al. (2013).

$$\text{Minimize} \quad \sum_{k \in R} c_k \sum_{t \in [0, \overline{d}-1]} o_{kt} + \rho \cdot \Delta\overline{T} \tag{1}$$

$$\text{subject to} \quad \sum_{t \in W_i} x_{it} = 1 \qquad i \in V \tag{2}$$

$$\sum_{t \in W_j} t \cdot x_{jt} - \sum_{t \in W_i} t \cdot x_{it} \geq \delta_{ij} \qquad \langle i, j \rangle \in E \tag{3}$$

$$x_{00} = 1 \tag{4}$$

$$o_{kt} \geq \sum_{i \in V} r_{kt} \sum_{\tau = \max\{ES_i; t - p_i + 1\}}^{\min\{t; LS_i\}} x_{i\tau} - Y_k \qquad k \in R,\ t \in T \tag{5}$$

$$\Delta\overline{T} \geq \sum_{t \in W_{n+1}} t \cdot x_{n+1,t} - \overline{T} \tag{6}$$

$$x_{it} \in \{0, 1\} \qquad i \in V,\ t \in W_i \tag{7}$$

$$o_{kt} \geq 0 \qquad k \in R,\ t \in T \tag{8}$$

$$\Delta\overline{T} \geq 0 \tag{9}$$

The objective function (1) to be minimized describes the overall cost depending on the total resource overload cost and the tardiness penalty. Constraints (2) and (7) express that for each activity $i \in V$ exactly one start time is applicable. Inequalities (3) ensure that all temporal constraints are satisfied and (4) sets the project start at time zero. Constraints (5) and (8) describe the resource overload of resource $k$ at time $t$. Finally, inequalities (6) and (9) define the delay of the prescribed project makespan, which cannot be negative.

In order to incorporate variable delay cost factors, the objective function (1) has to be revised to (12). Moreoever, constraints (6) and (9) have to be exchanged with (10) and (11), for which the applicable tardiness penalty $\overline{\rho}_\tau$ can be calculated beforehand for each feasible

$\tau \in W_{n+1}$.

$$\overline{\rho}_\tau := \sum_{t=0}^{\tau} \rho_t$$

Constraints (10) and (11) define a new auxiliary variable $\overline{\rho}$ that specifies the tardiness penalty of the chosen project makespan.

$$\overline{\rho} \geq \sum_{t \in W_{n+1}} \overline{\rho_t} \cdot x_{n+1,t} \tag{10}$$

$$\overline{\rho} \geq 0 \tag{11}$$

Since $\overline{\rho}$ already considers the tardiness penalty over all time units of delay, the objective function is adapted as described in (12).

$$\text{Minimize} \sum_{k \in R} c_k \sum_{t \in [0, \overline{d}-1]} o_{kt} + \overline{\rho} \tag{12}$$

Since the number of constraints (5) increases not only proportionally with the number of renewable resources, but also with the length of the planning horizon, the time discretization and maximum project duration is of importance.

## 5 Genetic algorithm

Motivated by population based approaches described in related work (Ballestin et al., 2007; Schnabel et al., 2018), e.g., a genetic algorithm appears promising for solving the resource overload problem with tardiness penalty. We propose a decoding procedure that only considers start times for each activity that result in quasistable schedules in regard to the tardiness penalty variant-dependent project subproblems. In addition, the genetic algorithm employs a representation that specifies for each activity a start time selection rule. To further improve the performance of the algorithm, we apply two learning methods: one to generate a diverse initial population and the other to analyse the best solutions of different generations in order to obtain adaptive mutation probabilities.

### 5.1 Representation

A schedule is encoded by a matrix with three rows and a column for each activity $i \in V \setminus \{0\}$. The genes of the project completion $n+1$ are of importance as they allow exploration of different subproblems given by their maximum project makespans, for instance $\overline{T}$. The first row of the representation contains random keys $rk_{1,i} \in [0, 1]$, which refer to the scheduling priority of activity $i$ (Kolisch & Hartmann, 1999). Preliminary tests have shown that using an activity list instead of priority values does not improve the performance of our genetic algorithm. The second value for each activity $i \in V \setminus \{0\}$ is a binary value $b_{2,i}$ specifying one of two start time selection rules. A value $b_{2,i} = 1$ signifies that an objective function value-oriented selection of start time $S_i$ is conducted. The other start time selection rule ($b_{2,i} = 0$) is based on an ascending order of possible start times $S_i$ for activity $i$. Starting from zero, each possible start time $S_i$ is assigned an equal sized subinterval of the domain of $rk_{3,i}$ according to the specified order. The domain of the random key $rk_{3,i}$ is chosen as $[0, 1)$ so that each subinterval has exactly the same length and the domain has a normalized length

of 1. For each activity $i$, $rk_{3,i}$ then lies exactly in one of the subintervals. The associated start time $S_i$ is then selected. The same technique for decoding $rk_{3,i}$ is utilized in Abbasi Iranagh (2015).

$$\begin{array}{c} \begin{array}{cccc} \text{activity 1} & \text{activity 2} & \ldots & \text{activity } n{+}1 \end{array} \\ \left[\begin{array}{cccc} rk_{1,1} & rk_{1,2} & \ldots & rk_{1,n+1} \\ b_{2,1} & b_{2,2} & \ldots & b_{2,n+1} \\ rk_{3,1} & rk_{3,2} & \ldots & rk_{3,n+1} \end{array}\right] \end{array} \begin{array}{l} \text{scheduling priority} \in [0, 1] \\ \text{start time selection rule} \in \{0, 1\} \\ \text{order-based selection} \in [0, 1) \end{array}$$

### 5.2 Initial population

The genetic algorithm starts with the generation of an initial population. It is realized in three steps using opposition-based learning to enhance the diversity of the initial population described in Rahnamayan et al. (2008). First, a number of solutions corresponding to the given population size is generated by randomly choosing the random key values from their respective interval with a uniform distribution. In contrast, we choose to set the binary values $b_{2,i}$ to 1 for all activities $i \in \{1, \ldots, n+1\}$. For each individual in the starting population, an individual is generated with opposite random keys in regards to its interval boundaries to create an opposite population. The binary values are still all set to 1. Finally, the initial population is composed by taking the best solutions of both sets until the given population size is reached. To further increase diversification, a newly generated solution whose schedule has already been obtained by another individual is always discarded.

### 5.3 Evolution

A new population is created using an elitist strategy where a fixed number of the best solutions from the population is directly inserted into the next one. The fitness value of an individual is set to be equal to the difference between the worst objective function value in the generation and its objective function value. To complete the new population, as many children are produced as necessary to maintain the population at the predetermined size. A pair of children is created by picking two parent solutions from the population with a roulette wheel selection, applying a uniform crossover and then a mutation operator. Afterwards, the parent pair is reinsert into the population. In our parametrized uniform crossover, it is determined for one child whether the set $\{rk_{1,i}, b_{2,i}, rk_{3,i}\}$ of each activity $i \in \{1, \ldots, n+1\}$ is inherited from the first parent or the second. Thereby, a predefined crossover probability gives the probability of selecting $\{rk_{1,i}, b_{2,i}, rk_{3,i}\}$ of the first parent. For the other child, the opposite choice is made for each activity $i \in \{1, \ldots, n+1\}$. Afterwards, a mutation operator is applied to both children considering a given mutation probability. Within the mutation operator, it is decided separately for each representation entry as to whether it is altered. For $rk_{1,i}$ and $rk_{3,i}$ a new value is drawn from the respective uniform distribution with a given probability or the current one is retained. For $b_{2,i}$ the current entry is changed to the other value with the same probability. It is possible to obtain children whose schedule already exists among the parent solutions or the children already created. To prevent premature convergence, these children are discarded.

To enable further learning, mutation probabilities for binary values $b_{2,i}$ are distinguished for each individual activity $i$ and the two possible values. These mutation probabilities are adjusted in the evolutionary process based on good solutions. When a new best solution is obtained, it is stored separately along with other solutions that were the best when they

first occurred. As soon as a predefined number $l$ of these solutions is acquired, they are evaluated. If for an activity $i$ the binary entry $b_{2,i}$ is the same across all solutions considered, the mutation probability for altering $b_{2,i}$ with this exact value is divided by a scaling factor $s$ greater than 1. The given mutation probability for the random keys remains unaffected. If, after that, a new best solution is found, the oldest considered solution is dismissed and the mutation probabilities for the binary values are again updated.

### 5.4 Decoding procedure

Algorithm 1 shows our decoding procedure, which is partly derived from schedule generation schemes described in Neumann and Zimmermann (1999) and Neumann et al. (2003). It starts with setting $S'_0 := 0$, resulting in a partial schedule $S'$. Moreover, the set of scheduled activities $C := \{0\}$, the set of unscheduled activities $\overline{C} := \{1, \ldots, n+1\}$ as well as the earliest and latest start times for every unscheduled activity are initialized (line 1–2). The decoding procedure exploits that a schedule $S$ is quasistable exactly if there is a spanning tree $T = \langle V, E^T, \delta^T \rangle$, where each arc $\langle i, j \rangle \in E^T$ represents either a binding temporal or precedence constraint. Thus, starting from $S'_0 := 0$, in each iteration the current partial spanning tree is extended by a corresponding edge connecting an activity $i \in \overline{C}$ to an activity $j \in C$, resulting in a quasistable partial schedule. For that purpose, a set of start times $D_i$ can be determined for every activity $i \in \overline{C}$. It consists of all start times $S_i \in \{ES_i, \ldots, LS_i\}$ for which there is at least one of the following constraints satisfied with at least one activity $j \in C$ and $S'_j \in S'$:

1. $S_i = S'_j + \delta_{ji}$ (temporal constraint)
2. $S_i = S'_j - \delta_{ij}$ (temporal constraint)
3. $S_i = S'_j + p_j$ (precedence constraint)
4. $S_i = S'_j - p_i$ (precedence constraint)

To additionally consider the quasistable schedules of all relevant subproblems, the decision set $D_{n+1}$ of the project completion $n+1$ is extended by the maximum project durations of all subproblems that are necessary according to the tardiness penalty variant.

5. $S_{n+1} = S'_0 + \overline{T}$ (constant delay cost factor)
6. $S_{n+1} = S'_0 + t, t \in \{\overline{T}, \ldots, \overline{d} - 1\}$ (variable delay cost factor)

In each scheduling step, an activity $h \in \overline{C}$ is eligible for scheduling if $D_h \neq \emptyset$ (line 7–13). Of the eligible activities, the activity $i \in \overline{C}$ with the highest $rk_{1,i}$ is chosen. Therefore, before determining $D_h$ for each activity $h \in \overline{C}$, it is examined first for each $h \in \overline{C}$ whether $rk_{1,h}$ exceeds the current highest priority value $rk_{1,max}$ in the respective iteration (line 6). If this is the case and $D_h \neq \emptyset$, the current highest random key $rk_{1,max}$ is updated and activity $h$ is kept as activity $i$ to be scheduled next. Otherwise, activity $h$ is discarded for this iteration. After all activities $h \in \overline{C}$ have been considered, the complete set $D_i$ of eligible start times is determined for the activity $i$ next to be scheduled (line 15–23). Once this set is established, the start time selection rule specified by binary value $b_{2,i}$ must be taken into account. If $b_{2,i} = 1$, the increase of the resource overload objective is calculated for every scheduling option $t \in \mathcal{D}_i$. Then, the start time $t$ with the smallest value is chosen as $S'_i$ (line 24). In the case where multiple $t \in \mathcal{D}_i$ result in the lowest resource overload function value, we choose the earliest start time among them. If $b_{2,i} = 0$, the interval $[0, 1)$ is divided into $m$ subintervals, where $m := |\mathcal{D}_i|$. The subinterval $[0, 1/m)$ is linked to the first entry in the ascending sorted set $\mathcal{D}_i$, the second to the subinterval $[1/m, 2/m)$ and so forth. Depending on which subinterval $rk_{3,i}$ lies in, the corresponding start time $S'_i$ is selected. The scheduling

process of an activity $i$ is completed when it is removed from the set of unscheduled activities $\overline{C}$ and inserted into the set of completed activities $C$. If $\overline{C}$ is empty, the algorithm terminates. The resulting schedule is always feasible.

**Algorithm 1** Decoding procedure

Set $S'_0 := 0$, initialize $C := \{0\}$ and $\overline{C} := V \setminus \{0\}$
Compute $ES_i := d_{0i}$ and $LS_i := -d_{i0}, \forall i \in \overline{C}$
**while** $\overline{C} \neq \emptyset$ **do**
  Initialize $i := -1$ and $rk_{1,max} := -1$
  **for** $h \in \overline{C}$ **do**
    **if** $rk_{1,h} > rk_{1,max}$ **then**
      **for** $j \in C$ **do**
        **if** $(\langle h, j\rangle \in E$ **and** $ES_h \leq S'_j - \delta_{hj} \leq LS_h)$
        **or** $(\langle j, h\rangle \in E$ **and** $ES_h \leq S'_j + \delta_{jh} \leq LS_h)$
        **or** $(ES_h \leq S'_j - p_h \leq LS_h)$
        **or** $(ES_h \leq S'_j + p_j \leq LS_h)$ **then**
          $i := h, rk_{1,max} := rk_{1,h}$.
          **break**
  Initialize $\mathcal{D}_i := \emptyset$
  **for** $j \in C$ **do**
    **if** $\langle i, j\rangle \in E$ **and** $ES_i \leq S'_j - \delta_{ij} \leq LS_i$ **then** $\mathcal{D}_i := \mathcal{D} \cup \{S'_j - \delta_{ij}\}$.
    **if** $\langle j, i\rangle \in E$ **and** $ES_i \leq S'_j + \delta_{ji} \leq LS_i$ **then** $\mathcal{D}_i := \mathcal{D} \cup \{S'_j + \delta_{ji}\}$.
    **if** $ES_i \leq S'_j - p_i \leq LS_i$ **then** $\mathcal{D}_i := \mathcal{D} \cup \{S'_j - p_i\}$.
    **if** $ES_i \leq S'_j + p_j \leq LS_i$ **then** $\mathcal{D}_i := \mathcal{D} \cup \{S'_j + p_j\}$.
    **if** $i = n + 1$ **and** constant delay cost **then**
      $\mathcal{D}_i := \mathcal{D}_i \cup (\{ES_{n+1}, \ldots, LS_{n+1}\} \cap \{\overline{T}\})$.
    **else if** $i = n + 1$ **and** variable delay cost **then**
      $\mathcal{D}_i := \mathcal{D}_i \cup (\{ES_{n+1}, \ldots, LS_{n+1}\} \cap \{\overline{T}, \ldots, \overline{d} - 1\})$.
  **if** $b_{2,i} = 1$ **then** choose $S'_i := \underset{\tau \in \mathcal{D}_i}{\operatorname{argmin}}(\Delta f_{ROP}(\tau))$
  **else** choose $\tau \in \mathcal{D}_i$ according to $rk_{3,i}$ and set $S'_i := \tau$
  Remove activity $i$ from $\overline{C}$, insert activity $i$ into $C$
  **for** $h \in \overline{C}$ **do**
    $ES_h := \max(ES_h, S'_i + d_{ih}), LS_h := \min(LS_h, S'_i - d_{hi})$

### 5.5 Lower bound of the optimal objective function value

The lower bound ($LB$) for the objective function value of the resource overload problem with tardiness penalty described in this subsection can be used as a termination criterion for the genetic algorithm or otherwise as an estimate of its solution quality. To compute $LB$, a lower bound for the cost of resource overload $\sum_{k\in\mathcal{R}} c_k o^{min}_{kS_{n+1}}$ can be estimated first for every feasible project makespan $S_{n+1} \in \{ES_{n+1}, \ldots, \overline{d}\}$, to which the applicable tardiness penalty $f_{TP}(S_{n+1})$ must then be added. Of these values, the minimum is then used as the lower bound for the optimal objective function value.

$$LB := \min_{S_{n+1}\in\{ES_{n+1},\ldots,\overline{d}\}} \left( \sum_{k\in\mathcal{R}} c_k o^{min}_{kS_{n+1}} + f_{TP}(S_{n+1}) \right)$$

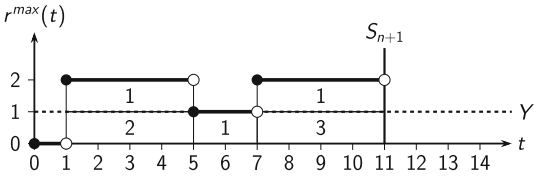


**Fig. 8** Resource utilization of all real activities at their feasible execution times for a project makespan of 12

There are several ways to determine the variables $o^{min}_{kS_{n+1}}$ for each $k \in \mathcal{R}$, of which the maximum is selected. An option is to calculate the total resource utilization of resource $k$ and reduce it by the product of the project duration $S_{n+1}$ and the resource threshold $Y_k$. This lower bound can be tightened by determining for each time period $t \in \{0, \ldots, S_{n+1} - 1\}$ the sum of resource units $r^{max}_k(t)$ utilized by all real activities that can possibly be in execution. If $r^{max}_k$ is below the threshold $Y_k$ in one or more periods, this may increase the resource overload that must occur in the remaining periods. A second way to determine the minimum resource overload $o^{min}_{kS_{n+1}}$ can be to look at the resource utilization $r_{ik}$ of each activity and evaluate whether and how much it exceeds $Y_k$ alone during its execution.

***Example 5*** We adjust the project network shown in Fig. 1 by adapting the minimum time lag between the project start 0 and activity 1 to $\delta_{01} = 1$, resulting in an earliest start $ES_1 = 1$. Moreover, the minimum and maximum time lag between the start of activity 2 and the start of activity 3 is increased to exactly 6. In this project, the three real activities have a total resource utilisation of 12 resource units. Let us consider a project makespan of $S_{n+1} = 11$. Calculating the minimum resource overload $o^{min}_{11}$ by reducing the total resource usage by the resource threshold $Y = 1$ multiplied by the project duration 11 yields in a value of 1. If we depict the maximum resource units $r^{max}$ possibly utilized in each time period (see Fig. 8), we see that in period 0 the maximum resource utilization is 0 and thus below the threshold $Y = 1$. The minimum resource overload $o^{min}_{11}$ therefore can be adjusted to 2. This is a tight lower bound for a project duration of 11 as the optimal resource overload function value equals 2 resulting, for example, from the schedule $S = (0, 5, 1, 7, 11)$. The other variant of determining $o^{min}_{kt}$ is not relevant here because none of the activities alone exceeds the resource threshold.

## 6 Experiments

In the following section, the performance of the MIP-formulation as well as the novel genetic algorithm is investigated. We first describe how we extend well-known instance sets to incorporate tardiness penalty. Then, the parametrization of the solution approaches is described. Finally, we present and discuss the results of experimental results for constant and variable delay cost, distinguishing between different prescribed project makespans and maximum project durations.

### 6.1 Test design

The computational tests are based on the test sets UBO for the RCPSP/max, which were obtained by the problem generator ProGen/max (Schwindt, 1998), cf). Every test set incorporates 90 instances with either 10, 20, 50 or 100 real activities and 5 renewable resources. In addition, the restrictiveness of Thesen (RT) is given for the instances that measures the degree to which the total number of feasible activity sequences is restricted by the precedence

relationships. If RT = 0, the real activities are all parallel to each other. RT = 1 occurs only for a serial project network. For the test instances, RTs in {0.25, 0.5, 0.75} are targeted.

In order to perform our experiments, further parametrizations of the instances are required. The maximum project duration is set with a coefficient $\alpha > 1$ to $\overline{d} := \lceil \alpha ES_{n+1} \rceil$. Moreover, the prescribed project makespan $\overline{T}$ is initialized as $\overline{T} := \lfloor \beta ES_{n+1} \rfloor$ where $1 \leq \beta < \alpha$. The resource thresholds $Y_k$ for all $k \in \mathcal{R}$ are chosen according to the total resource utilization and the prescribed project makespan $\overline{T}$ instead of the maximum project duration used in Neumann et al. (2003) ($Y_k := \left\lceil \sum_{i \in V} r_{ik} \cdot p_i / \overline{T} \right\rceil, k \in \mathcal{R}$). We assume that the cost factor $c_k$ for each overload unit of resource $k$ is 1. To calculate the tardiness penalty, a factor $\gamma$ is introduced. The constant delay cost factor $\rho$ is then calculated as $\rho := \gamma \sum_{k \in R} Y_k$. To test the solution approaches with variable delay cost factors, monotonically increasing cost factors $\rho_t$ are chosen.

$$\rho_t := \begin{cases} \gamma \cdot \sum\limits_{k \in R} Y_k \cdot (1+\gamma)^{(t-\overline{T})}, & t > \overline{T}; \\ 0, & t \leq \overline{T}. \end{cases}$$

We use C++ in the Microsoft Visual Studios 2022 development environment to implement the mixed-integer linear models and the genetic algorithm. For the MILPs the solver IBM ILOG CPLEX 20.1 is utilized. For CPLEX, the time limit is set to be 1800s for all instances. Preliminary results have shown that providing the previously described lower bound to the solver does not improve the run time or the best objective function value overall. The parameters of the genetic algorithm are chosen as follows: the population size is set according to the number of real activities in the instance to be $2n$. The proportion of elitist individuals is defined as 0.35 of the population size. The crossover probability is set to be 0.7 and the mutation probability as 0.1. For small instances with $n = 10$, the genetic algorithm runs until $K$-iterations without an improvement of the best objective function value are performed. We choose $K := 500$. For larger instances, a time limit is set. For instances with $n = 20$, the run time is limited to 30 s, for $n = 50$ and $n = 100$ to 300 s. The genetic algorithm is terminated early if the best objective function value equals the calculated lower bound. Preliminary runs have shown that the two learning methods (opposition-based learning and adapting mutation probabilities) lead to a better overall result for instances with 50 or more activities. Therefore, the learning methods are only applied for these instances. To create an initial population for smaller instances, only the first step described in Sect. 5.2 is performed. To adjust the mutation probabilities in case of $n = 50$ and $n = 100$, the number of analysed solutions is set to $l := 3$ and the scaling factor to $s := 4$.

For the first experiments, the maximum project duration is set using $\alpha := 1.25$. In order to examine the influence of different prescribed project makespans, we solve the instances with $\beta \in \{1.0, 1.1\}$. To determine $\gamma$, preliminary CPLEX runs are performed for instances with 10 activities, where optimality can always be proven in a short time for each instance solution. However, runs with $\gamma = 0.3$ for the resource overload problem with a constant delay factor and $\gamma = 0.1$ for variable delay cost factors have a slightly higher average run time in contrast to other parametrizations. Moreover, it is observed that the project durations of the optimal schedules are comparatively widely scattered across instances compared to other parametrizations. Therefore, $\gamma. = 0.3$ for the resource overload problem with a constant delay factor and $\gamma := 0.1$ for variable delay cost factors is investigated for all instance sizes.

**Table 2** Resource overload problem with a constant delay cost factor with $\alpha = 1.25$

| Instances | | | | CPLEX | | | Genetic algorithm | | |
|---|---|---|---|---|---|---|---|---|---|
| $\alpha$ | $\beta$ | $\gamma$ | $n$ | $\#opt$ | $\varnothing gap$ | $\varnothing t_{opt}[s]$ | $\varnothing gap_r$ | $\varnothing t_{ga}[s]$ | $\varnothing ss$ |
| 1.25 | 1.0 | 0.3 | 10 | 90 | 0 | 0.92 | 0.35 | 0.59 | 23.36 |
| | | | 20 | 79 | 1.17 | 149.71 | 0.76 | 30.00 | 19.90 |
| | | | 50 | 0 | 44.22 | −− | −6.68 | 300.02 | 20.72 |
| | | | 100 | 0 | 60.92 | −− | −13.42 | 300.14 | 17.55 |
| 1.25 | 1.1 | 0.3 | 10 | 90 | 0 | 1.00 | 0.14 | 0.81 | 23.61 |
| | | | 20 | 82 | 0.82 | 160.86 | 0.51 | 29.15 | 20.39 |
| | | | 50 | 0 | 38.62 | −− | −4.34 | 300.02 | 21.25 |
| | | | 100 | 0 | 60.16 | −− | −21.25 | 300.12 | 17.51 |

## 6.2 Results

All of the tests were conducted on computers with 64 GB RAM and an Intel(R) Core(TM) i7-7700K with 4x4.20 GHz and 8 threads. The results provide the number of instances per test set which are solved to optimality and its optimality is proven ($\#opt$) and the average gap over all instances ($\varnothing gap$) achieved by CPLEX. The average run time of the instances counted in $\#opt$ is denoted as $\varnothing t_{opt}$ and is measured in seconds. To investigate the quality of the genetic algorithm, the average relative gap ($\varnothing gap_r$), which is calculated in regards to the best objective function value obtained by CPLEX, is depicted. The average run time in seconds of the genetic algorithm is denoted as $\varnothing t_{ga}$. To obtain an estimate of the search space reduction, the number of start times within the decision set $|\mathcal{D}_i|$ is compared to the number of feasible start times $LS_i - ES_i + 1$ for each activity $i \in V\backslash\{0\}$ and the average over all solutions in all generations within a test set is denoted as $\varnothing ss$.

### 6.2.1 Constant delay cost

Table 2 summarizes the results for the resource overload problem with a constant delay cost factor, where we differentiate between two mentioned prescribed project makespan parametrizations, which also lead to different resource thresholds. For all of the instances with 10 activities, an optimal solution is found and its optimality is proven by CPLEX in under 11 s. Moreover, for the majority of instances with 20 activities (90 %) the solver determines an optimum within half an hour and $\varnothing t_{opt}$ is for both different prescribed project makespan parametrizations relatively equal. As we expected, the solution quality of the solver decreases with an increasing number of activities. CPLEX could not prove the optimality for any of the instances with 50 or 100 activities and the average gap is approximately 40% for $n = 50$ and 60% for $n = 100$ after half an hour of run time for both $\beta$. However, the gap varies significantly across the instances. The gap is considerably larger than the average for instances with a rather parallel network ($RT \approx 0.25$), whereby for more serial networks with $RT \approx 0.75$ the gap is significantly smaller. For the test sets with 50 or 100 activities, the correlation coefficients of RT and the solver gap lie in the range of $-0.8$ to $-0.7$.

The last columns of Table 2 reveal the performance results of the genetic algorithm. On instances with $n = 10$ and $n = 20$, the genetic algorithm performs similarly well as CPLEX. But especially for instances with 20 activities, the average run time $\varnothing t_{ga}$ is significantly shorter. An early termination due to reaching the calculated lower bound never occurs for

instances with a tight prescribed project makespan ($\beta = 1.0$). When setting $\beta = 1.1$, considering the calculated lower bound leads to the termination of the genetic algorithm 5 times with 10 activities and 3 times with 20 activities. All of the affected instances have a project duration that does not exceed the prescribed project makespan $\overline{T}$ and therefore no tardiness penalty applies. On larger instances with 50 activities, the genetic algorithm obtains an average relative gap of $-6.68\%$ and $-4.34\%$. Comparing the solution quality of instances with 100 activities, the genetic algorithm obtains on average significantly better results than CPLEX within a significantly shorter run time. With a tight prescribed project makespan ($\beta = 1.0$), CPLEX only obtains a better solution for 5 out of 90 instances. With $\beta = 1.1$ this is only the case for one instance.

Considering the average search space size $\varnothing ss$, it is evident that the genetic algorithm only considers a small subset of feasible start times and thus a search space reduction is in place. The values of $\varnothing ss$ for both prescribed project makespan parametrizations are similar, although no separate consideration of $\overline{T}$ as the project makespan is necessary for $\beta = 1.0$ in contrast to $\beta = 1.1$. In addition, we investigate the average of the binary values $b_{2,i}$ for all $i \in V \backslash \{0\}$ for the best solution obtained for each instance, in order to draw conclusions about the start time selection rules. The average varies widely for instances with 10 activities (0.25 to 1.0) and the least for instances with 50 and 100 activities (0.51 to 0.88). In regards to the good performance of the genetic algorithm, an objective function value-oriented start time selection, indicated by $b_{2,i} = 1$, seems to be appropriate for the resource overload problem with tardiness penalty. This is especially the case with medium-sized instances.

To test the usefulness of opposition-based learning and mutation probability adjustment, we conduct runs with 50 and 100 activities where we excluded the learning methods. The results show that for all four runs the gap is at least 0.9% higher without them, with the difference being greater for instance sets with 100 activities, being as high as 3.4%. With additional runs, we observed that the benefit of adapting mutation probabilities is far greater than the impact of opposition-based learning. We regard this as reasonable, since the latter affects only one generation.

### 6.2.2 Variable delay cost

In Table 3, the results of the experiments of the resource overload problem with variable delay cost factors parametrized with $\gamma = 0.1, \alpha = 1.25$ and $\beta \in \{1.0, 1.1\}$ are provided. Generally, the results shown are similar to the previous ones with a constant delay cost factor. Regarding small instances, the genetic algorithm terminates early for the same instances and the same $\beta$ as before. As the number of activities increases, the superiority of genetic algorithm is evident again. Regarding the test set with 100 activities and $\beta = 1.1$, the genetic algorithm obtains a better result than CPLEX for each of the 90 instances. For a tighter prescribed project makespan ($\beta = 1.0$), the genetic algorithm yields a better solution for 80 instances within 300 s. As expected, the average search space is larger than for the test sets with a constant delay cost factor. However, increasing $\beta$ from 1.0 to 1.1 results in a reduction of the number of project durations, which additionally need to be considered in the genetic algorithm. This search space reduction is also evident from $\varnothing ss$. The lowest average binary value for the best solution of the instances here is 33%, obtained for an instance with $n = 10$ and $\beta = 1.1$. Increasing the number of activities, leads also to an increase of the lowest average binary value found in the best solutions in both test sets up to 0.57. To us, this once again underlines the effectiveness of the two start time selection rules considered.

**Table 3** Resource overload problem with a variable delay cost factor with $\alpha = 1.25$

| Instances | | | | CPLEX | | | Genetic algorithm | | |
|---|---|---|---|---|---|---|---|---|---|
| $\alpha$ | $\beta$ | $\gamma$ | $n$ | $\#opt$ | $\varnothing gap$ | $\varnothing t_{opt}[s]$ | $\varnothing gap_r$ | $\varnothing t_{ga}[s]$ | $\varnothing ss$ |
| 1.25 | 1.0 | 0.1 | 10 | 90 | 0 | 1.36 | 0.34 | 0.60 | 27.21 |
| | | | 20 | 82 | 0.98 | 155.70 | 0.67 | 30.00 | 23.42 |
| | | | 50 | 0 | 43.57 | −− | −6.18 | 300.02 | 21.79 |
| | | | 100 | 0 | 59.49 | −− | −13.54 | 300.12 | 21.65 |
| 1.25 | 1.1 | 0.1 | 10 | 90 | 0 | 0.85 | 0.16 | 0.81 | 26.68 |
| | | | 20 | 80 | 0.98 | 93.88 | 0.80 | 29.01 | 22.23 |
| | | | 50 | 0 | 39.42 | −− | −6.20 | 300.02 | 21.62 |
| | | | 100 | 0 | 62.76 | −− | −28.75 | 300.13 | 18.94 |

**Table 4** Resource overload problem with a constant delay cost factor with $\alpha = 1.5$

| Instances | | | | CPLEX | | | Genetic algorithm | | |
|---|---|---|---|---|---|---|---|---|---|
| $\alpha$ | $\beta$ | $\gamma$ | $n$ | $\#opt$ | $\varnothing gap$ | $\varnothing t_{opt}[s]$ | $\varnothing gap_r$ | $\varnothing t_{ga}[s]$ | $\varnothing ss$ |
| 1.5 | 1.2 | 0.1 | 10 | 90 | 0 | 3.13 | 0.46 | 0.46 | 17.57 |
| | | | 20 | 63 | 2.57 | 349.73 | 1.09 | 26.03 | 14.93 |
| | | | 50 | 0 | 34.52 | −− | −11.34 | 300.02 | 15.57 |
| | | | 100 | 0 | 59.36 | −− | −32.82 | 300.14 | 12.62 |

### 6.2.3 Constant delay cost with a longer time horizon

We test the robustness of the solution approaches by increasing $\alpha := 1.5$ for instances with a constant delay cost factor (Table 4). We assume that with a longer planning horizon, the prescribed project makespan already allows more float. Therefore, we choose $\beta := 1.2$. To maintain a scattering of optimal project durations, $\gamma$ is reduced to 0.1. With the increased planning horizon, both approaches deteriorate on instances with 10 and 20 activities, with CPLEX being more affected especially regarding the number of optimal solutions found and its optimality proven as well as $\varnothing t_{opt}$. For $n = 10$, the genetic algorithm terminates 16 times due to reaching the calculated lower bound, which explains the lowest average run time for $n = 10$ over all experiments. For $n = 20$, the genetic algorithm terminates 9 times early. The respective instances for both numbers of activities have again a project duration equal or less than $\overline{T}$. Across all experiments, the best average relative gaps ($\varnothing gap_r$) are obtained for problem instances with 50 and 100 activities. Since the number of start times considered in the genetic algorithm depend primarily on the number of other activities and not on the planning horizon in contrast to the mathematical model formulation, this performance improvement seems reasonable to us. In addition, for 83 instances containing 100 real activities the calculated lower bound is better that the one provided by CPLEX. If this lower bound is considered, the average gap for the genetic algorithm is 40.46% and thus nearly 20% lower than the average gap of CPLEX.

## 7 Conclusion

In this paper, we have shown that the set of quasistable schedules can be adapted to reduce the search space for the resource overload problem with tardiness penalty. Besides a mixed-integer linear model, we present a genetic algorithm with a decoding procedure that ensures that only solutions within the reduced search space are examined. The experiments show that small instances are solved efficiently by CPLEX. Our genetic algorithm performs similarly well as the solver on these instances, however on average it is slightly faster. On the majority of the medium sized instances, the devised genetic algorithm outperforms the mixed-integer linear models implemented in CPLEX. An area of future research could be the development of an exact algorithm for the resource overload problem with tardiness penalty. To further enable a trade-off between resource levelling and the project makespan, a multi-mode resource overload problem with tardiness penalty can be investigated. Of particular interest are activity execution modes that lead to different processing times (Węglarz et al., 2011). Both constant resource utilizations for each execution mode and varying execution intensities in terms of a workload perspective can be considered (Bianco et al., 2016; Tarasov et al., 2021). Future research could address, to what extent the search space can be reduced for these extensions and which prerequisites that have to be satisfied for that.

**Funding** Open Access funding enabled and organized by Projekt DEAL. No funding was received to assist with the preparation of this article

## Declarations

**Conflicts of interest** There are no interests to declare.

**Ethical approval** This article does not contain any studies with human participants or animals performed by any of the authors.



## References

Abbasi Iranagh, M. (2015). *Development of high performance heuristic and meta-heuristic methods for resource optimization of large scale construction projects*. Ph.D. thesis, Middle East Technical University.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows*. Prentice Hall.

Atan, T., & Eren, E. (2018). Optimal project duration for resource leveling. *European Journal of Operational Research, 266*(2), 508–520.

Ballestin, F., Schwindt, C., & Zimmermann, J. (2007). Resource leveling in make-to-order production: Modeling and heuristic solution method. *International Journal of Operations Research, 4*(1), 50–62.

Bianco, L., Caramia, M., & Giordani, S. (2016). Resource levelling in project scheduling with generalized precedence relationships and variable execution intensities. *OR Spectrum, 38*, 405–425.

Gerhards, P., & Stürck, C. (2018). A hybrid metaheuristic for the multi-mode resource investment problem with tardiness penalty. In A. Fink, A. Fügenschuh, & M. J. Geiger (Eds.), *Operations Research Proceedings 2016* (pp. 515–520). Cham: Springer.

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.

Hegazy, T. (1999). Optimization of resource allocation and leveling using genetic algorithms. *Journal of Construction Engineering and Management, 125*(3), 167–175.

Kim, J., Kim, K., Jee, N., & Yoon, Y. (2005). Enhanced resource leveling technique for project scheduling. *Journal of Asian Architecture and Building Engineering, 4*(2), 461–466.

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In J. Węglarz (Ed.), *Project scheduling* (pp. 147–178). Kluwer.

Koulinas, G. K., & Anagnostopoulos, K. P. (2012). Construction resource allocation and leveling using a threshold accepting-based hyperheuristic algorithm. *Journal of Construction Engineering and Management, 138*(7), 854–863.

Kreter, S., Rieck, J., & Zimmermann, J. (2014). The total adjustment cost problem: Applications, models, and solution algorithms. *Journal of Scheduling, 17*(2), 145–160.

Neumann, K., Schwindt, C., & Zimmermann, J. (2003). *Project scheduling with time windows and scarce resources* (2nd ed.). Springer.

Neumann, K., & Zimmermann, J. (1999). Resource levelling for projects with schedule-dependent time windows. *European Journal of Operational Research, 117*(3), 591–605.

Ponz-Tienda, J. L., Yepes, V., Pellicer, E., & Moreno-Flores, J. (2013). The resource leveling problem with multiple resources using an adaptive genetic algorithm. *Automation in Construction, 29*(2), 161–172.

Pritsker, A. A. B., Waiters, L. J., & Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science, 16*(1), 93–108.

Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. A. (2008). Opposition-based differential evolution. *IEEE Transactions on Evolutionary computation, 12*(1), 64–79.

Rieck, J., & Zimmermann, J. (2015). Exact methods for resource leveling problems, In C. Schwindt & J. Zimmermann (Eds.), *Handbook on Project Management and Scheduling* (Vol. 4, pp. 361–387). Springer.

Rieck, J., Zimmermann, J., & Gather, T. (2012). Mixed-integer linear programming for resource leveling problems. *European Journal of Operational Research, 221*(1), 27–37.

Schnabel, A., Kellenbrink, C., & Helber, S. (2018). Profit-oriented scheduling of resource-constrained projects with flexible capacity constraints. *Business Research, 11*(2), 329–356.

Schwindt, C. (1998). Generation of resource-constrained project scheduling problems subject to temporal constraints. Technical Report WIOR-543, Institute for Economic Theory and Operations Research, University Karlsruhe.

Shadrokh, S., & Kianfar, F. (2007). A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. *European Journal of Operational Research, 181*(1), 86–101.

Tarasov, I., Hait, A., & Battaia, O. (2021). Benders decomposition for a period-aggregated resource leveling problem with variable job duration. *Computers & Operations Research, 132*, 105258.

Węglarz, J., Józefowska, J., Mika, M., & Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes-a survey. *European Journal of Operational Research, 208*(3), 177–205.