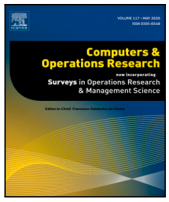
Computers & Operations Research 161 (2024) 106453

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



# $\Gamma$-robust optimization of project scheduling problems

Arie M.C.A. Koster, Jenny Segschneider *, Nicole Ventsch

*RWTH Aachen University, Research Area Discrete Optimization, Pontdriesch 10-12, 52056, Aachen, Germany*

ARTICLE INFO

*Keywords:*
Robust project scheduling
Budgeted uncertainty
Two-stage robust optimization
Heuristics
Mixed-integer programming

ABSTRACT

In this paper, we investigate the problem of finding a robust baseline schedule for the project scheduling problem under uncertain process times. We assume that the probability distribution for the duration is unknown, but an estimate is given along with an interval in which this time can vary. At most $\Gamma$ of the tasks will deviate from the estimated time.

We present two approaches to solving this problem. The first approach treats the problem of determining the earliest guaranteed finish times and can be solved in polynomial time by an extension of the critical path method. The second is a two-stage approach that determines the baseline schedule in the first stage and an adaptation to the scenario in the second stage. We show strong NP-hardness of the second stage problem and introduce a novel formulation. From this formulation we derive an exact algorithm and three heuristics. A computational study on benchmark instances shows that the heuristics perform well on larger instances.

## 1. Introduction

The classical project scheduling problem (PSP) handles the scheduling of a set of tasks within a project such that the makespan, which is the total time needed, is minimal. It is an important subject in operations research and has been widely studied in the past. We refer to Fahmy (2016) for an overview of different models and algorithms solving PSP. In particular, the resource-constrained PSP has been investigated by many. Originally, PSP is defined as a deterministic problem without uncertainties. In real life, the problem parameters in planning problems are typically uncertain. To gain a better understanding of the impact of such uncertainties, we consider in this paper uncertain task durations. We focus on the PSP without resource constraints as the resource-constrained PSP is already NP-hard. Most of the ideas presented in this paper can be transferred to the resource-constrained case. If all parameters are known with certainty, PSP can be solved easily using the critical path method introduced by Kelley and Walker (1959). It is common practice to estimate times needed for each task without considering the underlying uncertainty. These estimates are often imprecise and the finish times determined by the critical path method can often not be met in practice.

Recently, more research efforts aimed at modeling the reality more precisely by taking uncertainty into consideration. Herroelen and Leus (2005) identified six main approaches to solve these problems: reactive scheduling, stochastic scheduling, GERT network scheduling, fuzzy scheduling, sensitivity analysis and the approach that is used in this paper, robust scheduling. These approaches have some basic differences. For example, using reactive scheduling results in a schedule without uncertainties, but during the project execution it is adjusted if necessary. Opposite to that, in stochastic project scheduling usually no initial schedule is created, but decisions are made at multiple time points throughout the project based on observations of the past and available a-priori information. The approach we have chosen in this paper, robust scheduling, aims at creating an initial schedule which is protected against uncertain events that can occur during the project execution. This is especially important for projects where delay of some or all tasks would come at a very high cost. For example, when bidding for large-scale construction projects, construction companies may overpromise to increase their chances of winning the contract. However, failing to meet promised deadlines can be costly in both reputation and money.

In robust optimization, scenarios from a predefined uncertainty set are considered. For a survey on robust optimization, see for example Ben-Tal et al. (2009). In this paper, we will consider so-called budgeted uncertainty, also known as $\Gamma$-robustness, proposed by Bertsimas and Sim (2004) for two robust variants of the PSP, the single stage and the two-stage problem.

The $\Gamma$-robust single-stage PSP treats the problem of determining the earliest guaranteed finish time for every task. The finish times have to be feasible for all scenarios within the uncertainty set, so that the times can be kept in every possible scenario. As single-stage robust optimization is known to produce very conservative solutions at the

* Corresponding author.
*E-mail addresses:* koster@math2.rwth-aachen.de (A.M.C.A. Koster), segschneider@math2.rwth-aachen.de (J. Segschneider).

https://doi.org/10.1016/j.cor.2023.106453
Received 19 December 2022; Received in revised form 6 October 2023; Accepted 7 October 2023
Available online 10 October 2023

expense of the objective value, we additionally propose a $\Gamma$-robust two-stage problem based on the stochastic approach from Zhu et al. (2007). They suggest a model which balances the cost of the baseline schedule and the expected cost for deviations from this schedule using stochastic scheduling. However, due to the unique nature of most projects, a probability distribution is rarely known. Sometimes the project has never been executed before, so there is no empirical information available. For other projects, minimizing the expected costs, but having a positive probability of very bad results is unacceptable. In the example above, when bidding for large scale construction projects, too high costs can lead to the bankruptcy of the construction company, as has happened several times in the construction of the Berlin Brandenburg Airport (Berlin, 2020). Another example is project scheduling problems in the medical field, where high costs in an "unlikely" scenario could result in loss of lives. As a countermeasure, we employ robust scheduling to determine the schedule with the lowest cost in the worst-case scenario.

Our two-stage formulation is similar to the concept of recoverable robustness proposed by Liebchen et al. (2009): the first stage consists of setting a baseline schedule that is adapted to the worst-case scenario in the second stage. The objective is to minimize the cost of the baseline schedule plus the adaptation costs in the worst-case. This approach allows for a more flexible schedule while still protecting against uncertainties.

The contributions of this paper are manyfold. First, we introduce a new derivation for the $\Gamma$-robust Single-Stage PSP which can be solved using the algorithm from Bruni et al. (2017). They found a similar problem with a different objective in the subproblem of their two-stage formulation. They introduced a polynomial time algorithm based on dynamic programming which can also be used to solve our single-stage PSP. The same problem was further addressed by Bold and Goerigk (2021) who proposed a compact LP formulation based on the algorithm.

As a second contribution, we propose a robust approach for the stochastic model of Zhu et al. (2007) and create a novel formulation for the recoverable robust PSP. In the process of this, we also show NP-hardness of the resulting subproblem. For the resulting problem, an exact algorithm and three heuristics are proposed and tested in a computational study. Finally, we outline the changes needed to adapt the exact algorithm and two of the three heuristics to the resource-constrained project scheduling problem.

This paper is organized as follows. In the next section, we give a brief literature review over variants of the robust two-stage PSP. In Section 3, we introduce the single-stage project scheduling problem and compare it to the problem introduced by Bruni et al. (2017). Subsequently, we will revisit their findings and transfer them to the single-stage PSP. Section 4 treats the $\Gamma$-robust two stage problem. Again, the problem is first defined formally and then a reformulation using vertex enumeration is introduced. From this, we derive an algorithm which can solve the problem exactly but needs exponential running time. Since this algorithm is not applicable for large projects, we will also derive a heuristic which uses McCormick Envelopes (Tsoukalas and Mitsos, 2014) and gives us an upper bound for the optimal solution. Moreover, we develop a polynomial heuristic based on the single-stage algorithm. In Section 5, we then test the three algorithms by conducting a computational study. Finally, we provide a conclusion and a short outlook in Section 5. Most results have been presented first in the Master's thesis of the last author (Ventsch, 2020).

## 2. Literature review

Robust two-stage optimization approaches have been rarely used for PSP, primarily in resource-constrained project scheduling. A recent overview was given by Hazır and Ulusoy (2020). For example, Artigues et al. (2013) developed a scenario-relaxation algorithm minimizing the worst-case absolute regret and were the first to apply robust optimization to PSP. Furthermore, Bruni et al. (2015) proposed a chance-constraint based heuristic and a two-stage adjustable robust optimization model (Bruni et al., 2017). The first stage in their two-stage model consists of sequencing decisions that determine the order of some project tasks to ensure resource constraints, followed by the schedule in the second stage. They (Bruni et al., 2018) also compare this algorithm to an additional Benders-style algorithm in a computational study. Recently, Bold and Goerigk (2021) introduced a compact reformulation of the two-stage robust resource-constrained PSP with resource allocation decisions made in the first stage and scheduling decisions in the second stage. Lately, these approaches have been applied to the robust multi-mode resource-constrained PSP, where each task can be performed in different ways requiring different time and resources. Balouka and Cohen (2019) proposed a two-stage model for this problem and solve it using a Benders decomposition approach. In the first stage, mode selection and resource allocation is determined and in the second stage, the final schedule is made. Most recently, Bold and Goerigk (2022) improved on their Benders Decomposition approach and proposed a compact formulation for the robust multi-mode resource-constrained PSP. These approaches all assume that the time schedule can be made after realization of the scenario as long as the resources are allocated accordingly. However, this is not always the case. In some cases, a baseline schedule is necessary for planning and deviating from it can lead to high costs.

Furthermore, the Dynamic Scheduling problem of fixing a given schedule that has become infeasible due to a time or resource disruption has been studied recently. The focus of this problem is on the response to the disruption and, in contrast to our work, not on the specific disruption. Additionally, in most cases the baseline schedule is assumed as given, whereas we can set the baseline schedule to get a better starting position in this problem. To solve the Dynamic Scheduling Problem, Sakkout and Wallace (2000) use probe backtracking. Artigues and Roubellat (2000) propose a polynomial time algorithm for the problem of adding tasks into a solution of the multi-mode resource-constrained PSP. Bendotti et al. (2017) examine the anchored resource-constraint PSP, in which a partial baseline schedule is adapted to disruption while changing as few tasks as possible. Finally, Zaman et al. (2021) propose a pro-reactive approach for the multi-mode resource-constrained project scheduling problem with two stages. In the first stage, they set an initial schedule which keeps free resources over the entire project, which are used in the second stage as a buffer for any disruptions.

Moreover, there are different approaches often referred to as robust. Herroelen and Leus (2004a) developed a linear programming formulation with the objective of finding a baseline schedule that minimizes the expected weighted deviation of the starting times in the realized schedule from the baseline schedule. Another approach of calculating such a schedule using fuzzy scheduling is shown in Ke and Liu (2010) in which the fuzzy times are simulated and the resulting instances are used to determine schedules using the critical path method. A review of other results is given by Herroelen and Leus (2004b).

## 3. The $\Gamma$-robust single-stage problem

The deterministic PSP consists of a directed, acyclic graph $G = (V, A)$ and durations $(t_v)_{v \in V}$. The nodes in $V = \{v_0, \dots, v_{n+1}\}$ each correspond to a task which takes $(t_v)_{v \in V}$ time to be completed. The nodes $v_0$ and $v_{n+1}$ are dummy source and sink with duration 0 representing the start and end of the project. The arcs denote the predecessor relationship. A task $w \in V$ can only be started after every task $v \in V$ with $(v, w) \in A$ is finished.

### 3.1. Budget uncertainty

For the $\Gamma$-robust Single-Stage Problem, we now assume that the duration of each activity $v \in V$ is unknown and lies somewhere between its estimation $t_v$ and its worst-case value $t_v+u_v$. Furthermore, we assume that at most $\Gamma$ tasks will not finish in the estimated time following the ideas by Bertsimas and Sim (2004). Therefore, the so-called uncertainty set is given by

$$\mathcal{U}_\Gamma := \{\xi \in \mathbb{R}_+^{|V|} : \|\xi\|_\infty \leq 1,\ \|\xi\|_1 \leq \Gamma\}$$

and the time needed for all tasks $v \in V$ in a scenario $\xi \in \mathcal{U}_\Gamma$ is given by $t_v(\xi) = t_v + \xi_v \cdot u_v$. Additionally, we set $t_{v_0}(\xi) = t_{v_{n+1}}(\xi) = 0$ for each scenario.

The parameter $\Gamma$ controls the robustness of the solution. For $\Gamma = 0$, no task can be delayed and we solve the deterministic PSP. Conversely, for $\Gamma = n$ each task can be delayed simultaneously. Thus, a higher $\Gamma$ will lead to a more conservative optimal solution with worse optimal value but better robustness against uncertainty. Straightforward adaptation of the critical path method is not easy as the uncertainty is part of the right hand side in the LP-formulation (Minoux, 2011).

Instead, our goal is to determine the earliest guaranteed finish time $F_v$ for each task $v \in V$ which is feasible in every scenario. In order to formally define the problem, we use $P_v$ to denote the set of all paths from $v_0$ to $v \in V$. A finish time for task $v$ can be guaranteed in every scenario if the duration of every path from $v_0$ to $v$ is smaller than $F_v$. Therefore, the problem can be described as follows:

$$\begin{aligned} \min\ & \sum_{v\in V} F_v \\ \text{s.t. } & F_{v_0} = 0 \\ & F_v \geq \sum_{w\in p_v} \left(t_w + u_w \cdot \xi_w\right) \qquad \forall v \in V,\ p_v \in P_v,\ \xi \in \mathcal{U}_\Gamma \end{aligned} \tag{1}$$

Alternatively, a weighted sum of the finish times can be minimized. Since there are possibly exponentially many constraints, a cutting plane type algorithm seems appropriate. Hence, we will now analyze the separation problem of finding a maximal path from $v_0$ to $v'$ and thus a (possibly) violated constraint for a given solution $F_v$ and a node $v'$. We call this path a violated path. That way, we get the earliest guaranteed finish time for the task $v'$ by only considering the critical path. For each task $v \in V$ and arc $a \in A$, let

$$x_v := \begin{cases} 1 & \text{task } v \text{ on critical path} \\ 0 & \text{task } v \text{ otherwise} \end{cases} \quad \text{and}$$

$$z_a := \begin{cases} 1 & \text{arc } a \text{ on critical path} \\ 0 & \text{arc } a \text{ otherwise.} \end{cases}$$

Our aim is to find a violated inequality, i.e. a violated path, and a scenario, such that the total time of a single path from $v_0$ to $v'$ is maximal. Thus, we want to solve the following Mixed Integer Program:

$$\max \sum_{v\in V} \left(t_v x_v + u_v \xi_v\right) \tag{2a}$$

$$\text{s.t.} \sum_{a=(v,\cdot)\in A} z_a - \sum_{a=(\cdot,v)\in A} z_a = 0 \qquad \forall v \in V\backslash\{v_0, v'\} \tag{2b}$$

$$x_v = \sum_{a=(\cdot,v)\in A} z_a \qquad \forall v \in V\backslash\{v_0\} \tag{2c}$$

$$x_{v_0} = x_{v'} = 1 \tag{2d}$$

$$\sum_{v\in V} \xi_v \leq \Gamma \tag{2e}$$

$$\xi_{v_0} = \xi_{v_{n+1}} = 0 \tag{2f}$$

$$\xi_v \leq x_v \qquad \forall v \in V \tag{2g}$$

$$\xi \in [0,1]^{|V|},\ x \in \{0,1\}^{|V|},\ z \in \{0,1\}^{|A|} \tag{2h}$$

The objective (2a) is to maximize the estimated time of the path defined by $x$ together with the delay through uncertainty on the entire graph. Since, according to (2g), delays can only occur for tasks on the violated path, this equals the time needed for the violated path and therefore the earliest finish time for $v'$ in the worst-case scenario. The constraints (2b)–(2d) model a unit flow from $v_0$ to $v'$. Constraints (2b) are flow-conservation constraints, (2c) ensures the definition of $x$ and (2d) sets $v_0$ and $v'$ as first and last node of the violated path. Altogether, these constraints ensure that $x$ defines a path from $v_0$ to $v'$. Furthermore, constraints (2e) and (2f) deal with the $\Gamma$-robustness as defined above.

The violated paths resemble the critical paths defined by, among others, Kelley and Walker (1959). At this point, it is important to notice that the graph is acyclic and thus no subtours are possible. Therefore, the ILP will always have an optimal solution. If the optimal solution value is larger than $F_v$, then a violated inequality is found.

In the remainder of this section, we will compare this approach to a similar one found in the literature and present a polynomial-time algorithm to solve it.

### 3.2. Relation to RCPSP

A similar problem was first examined by Bruni et al. (2017, Section 5). They introduce a new formulation for the two stage resource-constrained PSP (TSRCPSP), where the subproblem resembles our formulation, the only difference being the objective function. After being rewritten and linearized in Section 5, the resulting formulation is equivalent to the MIP presented above. They also propose an algorithm based on dynamic programming that solves the problem in polynomial time. We will now compare the two problems, revisit their algorithm, and show that the solution found by the algorithm for $v' = v_{n+1}$ maximizes both the makespan $F_{v_{n+1}}$ and the finish time for each node. Therefore, the algorithm solves the original problem (1). Subsequently, in the following, we will only consider the case $v' = v_{n+1}$.

To this end, we will now briefly review the problem investigated by Bruni et al. (2017). In the resource-constrained PSP (RCPSP), each task additionally requires an amount $r_{vk} \in \mathbb{Z}_+$ of resource $k$ from a given set of resources $K$. In each time period and for each resource $k \in K$, the amount of consumed resource in a solution cannot exceed the finite availability $R_k$. We call a set $C \subseteq V$ a forbidden set if for at least one resource $k \in K$ it holds $\sum_{v\in C} r_{vk} > R_k$ and therefore the resource constraint is violated. Additionally, it is called minimal if none of its subsets are forbidden sets. The Main Representation Theorem of Bartusch et al. (1988) states that a solution for the RCPSP can be reduced to extending $G$ by a set of extra arcs $X \subseteq (V \times V)\backslash A$ such that $G' = (V, A \cup X)$ is acyclic and the transitive closure of $A \cup X$ has no minimal forbidden sets. The set $X$ is called a sufficient selection. This leads to a two-stage formulation where in the first stage a sufficient selection is chosen and in the second stage the schedule is computed. The objective is to find a sufficient selection that minimizes the worst case makespan under uncertainty. For a sufficient selection $X$, the extended Graph $G'$ is represented by the binary variable $y \in \{0,1\}^{|V\times V|}$ with $y_{vw} = 1$ iff $(v,w) \in E \cup X$. Let $y$ be the binary (fixed) variable representing the extended Graph for a fixed sufficient selection $X$. Then the second stage problem is given by

$$\begin{aligned} \max_{\xi\in\mathcal{U}_\Gamma} \min_{(F_v)_{v\in V}}\ & F_{n+1}(\xi) \\ \text{s.t. } & F_0 = 0 \\ & F_w(\xi) - F_v(\xi) \geq t_v + \xi_v \cdot u_v - M \cdot (1 - y_{vw}) \qquad \forall (v,w) \in V \times V \\ & F_v(\xi) \geq 0 \qquad \forall v \in V \end{aligned}$$

where $M$ is large enough.

It is similar to (2) with the two main differences lying in the objective. The objective of the subproblem presented by Bruni et al. is to find the worst case makespan after realization. In contrast to our

formulation (1), the makespan is minimized instead of the sum of all finish times and the schedule is computed depending on the scenario $\xi$. However, using a strong duality result for the inner minimization problem leads to the following formulation.

$$\begin{aligned}
\max_{z,\xi} \quad & \sum_{(v,w)\in A\cup X} (t_v + \xi\cdot u_v)\cdot z_{(v,w)} - M\cdot \sum_{(v,w)\in V\times V\setminus(A\cup X)} (t_v+\xi\cdot u_v)\cdot z_{(v,w)} \\
\text{s.t.} \quad & \sum_{a=(\cdot,v_{n+1})\in A} z_a = 1 \\
& \sum_{a=(v_0,\cdot)\in A} z_a = 1 \\
& \sum_{a=(v,\cdot)\in A} z_a - \sum_{a=(\cdot,v)\in A} z_a = 0 && \forall\, v\in V\setminus\{v_0,v_{n+1}\} \\
& \sum_{v\in V} \xi_v \le \Gamma \\
& z_a \in \{0,1\} && \forall\, a\in A \\
& 0\le \xi_v \le 1 && \forall\, v\in V
\end{aligned}$$

This formulation describes the same problem as (2a)–(2h) on the extended Graph $G' = (V, A\cup X)$. Since $M$ is very large, in an optimal solution, the right sum in the objective and therefore $z_{(v,w)}$ for each $(v,w)\in V\times V\setminus(A\cup X)$ will be zero. The only differences lie in the additional variables $(x_v)_{v\in V}$ and the constraint (2g) which leads to a linear objective function (2a). Therefore, the following algorithm proposed by Bruni et al. (2017) can also be applied to solve our problem.

### 3.3. Dynamic program

Alternatively, a dynamix program to solve (1) can be derived and runs in polynomial time. For each node $v\in V$ it considers $\Gamma+1$ paths $p_v^0,\dots,p_v^\Gamma$ from the source node to $v$, where each path $p_v^\gamma$ includes exactly $\gamma\le\Gamma$ delays. For a node $v_j$, the longest path with exactly $\gamma$ delays is evaluated by considering two possibilities: either a predecessor task $j$ is delayed resulting in $p_{v_i}^\gamma = p_{v_j}^{\gamma-1}\cup\{v_i\}$, or it is not delayed resulting in $p_{v_i}^\gamma = p_{v_j}^\gamma\cup\{v_i\}$.

We denote the duration of the longest path from $v_0$ to $v_j$ with exactly $\gamma$ delays as $F_{v_j,\gamma}$. These considerations lead to the following recursive algorithm.

$$\begin{aligned}
& F_{v_0,\gamma} = 0 && \forall\,\gamma = 0,\dots,\Gamma \\
& F_{v_j,0} = \max_{i:(v_i,v_j)\in A}\left\{F_{v_i,0}+t_{v_j}\right\} && \forall\, v_j\in V\setminus\{v_0\} \\
& F_{v_j,\gamma} = \max_{i:(v_i,v_j)\in A}\left\{\max\left(F_{v_i,\gamma}+t_{v_j}, F_{v_i,\gamma-1}+t_{v_j}+u_{v_j}\right)\right\} && \forall\, \begin{matrix} v_j\in V\setminus\{v_0\},\\ \gamma=1,\dots,\Gamma\end{matrix}
\end{aligned} \tag{3}$$

This recursion is well-defined since the Graph is acyclic and for each arc $(v_i,v_j)\in A$ it holds $i<j$. Since the results $F_{v,\Gamma}$ for each node $v\in V$ are independent of the sink $v' = v_{n+1}$, the finish times $F_{v,\Gamma}$ can be reached in each scenario and the schedule $(F_{v,\Gamma})_{v\in V}$ is an optimal solution for the original problem (1).

The runtime of this algorithm is $\mathcal{O}(\Gamma\cdot|V|^2)$ because the number of states $F_{v,\gamma}$ is $(\Gamma+1)\cdot|V|$ and the computation of each state, in the worst-case, takes $\mathcal{O}(|V|)$ operations. Since $\Gamma\le|V|$, we know that the complexity is at most $\mathcal{O}(|V|^3)$, so that this algorithm runs in polynomial time.

## 4. The $\Gamma$-robust two-stage problem

We will now extend our problem so that it can represent another aspect of the project scheduling problem. In real projects, it is usually attempted to maximize the profit and minimize the costs. The costs of projects heavily depend on the initial project schedule as well as on deviations from the set baseline schedule in case it needs to be adjusted after the project already started. This can result in higher personnel costs, penalty fees or in costs for advance delivery of necessary material. Thus, we will now model the problem as a two-stage approach where the estimated project schedule is determined in a way that it minimizes the costs for the initial project schedule together with the costs for deviating from this schedule in the worst case scenario. By this definition, every schedule $(F_v)_{v\in V}$ is feasible but may deviate heavily depending on the scenario. This approach is based on the stochastic model presented by Zhu et al. (2007). Note, that the following results can also be transferred to the resource-constrained PSP. In this case, the resource allocation decisions would be made together with the baseline schedule prior to the realization of the scenario by setting a sufficient selection $X$ (cf. Section 3.2) in the first stage. The second stage problem would then be solved on the expanded graph $G=(V,A\cup X)$.

Additionally to the notation used in Section 3, we need the following definitions:

- $c_v\ge 0$ is the cost for task $v\in V$ per time unit; we set $c_{v_0} := c_{v_{n+1}} := 0$
- $y_v^+\ge 0$ is the number of time units that the finish time of task $v\in V$ is later than in the formerly set project schedule $(F_v)$
- $y_v^-\ge 0$ is the number of time units that the finish time of task $v\in V$ is earlier than in the formerly set project schedule $(F_v)$
- $d_v^+\ge 0$ is the cost per time unit for finishing task $v\in V$ later than in the formerly set project schedule $(F_v)$
- $d_v^-\ge 0$ is the cost per time unit for finishing task $v\in V$ earlier than in the formerly set project schedule $(F_v)$

Our goal is to minimize the costs associated with the project schedule, which are given by $c^TF = \sum_{v\in V} c_v\cdot F_v$, together with the deviation penalty for this schedule. For each task $v\in V$, the new finish time including deviation penalty is given by $F_v + y_v^+ - y_v^-$. Assuming that the schedule $F$ and the scenario $\omega$ are known, the deviations $y_v^+, y_v^-\ge 0$ have to satisfy the following condition:

$$F_w + y_w^+(\omega) - y_w^-(\omega) \ge F_v + y_v^+(\omega) - y_v^-(\omega) + t_w(\omega) \quad \forall\,(v,w)\in A$$

This ensures that for nodes with a predecessor relation the new finish time for task $w$ will never be earlier than the new finish time for task $v$ plus the time $t_w(\omega)$ needed to complete task $w$ in the current scenario. Moreover, the artificial start node $v_0$ cannot deviate from the schedule, so that we have $y_{v_0}^+ = y_{v_0}^- = 0$.

The costs for these deviations need to be minimized, so that we in total get the following linear program (where scenario $\omega$ is defined by vector $\xi$):

$$\begin{aligned}
f(\xi,F) := \min_{y^+,y^-} \quad & \sum_{v\in V} d_v^-\cdot y_v^- + d_v^+\cdot y_v^+ \\
\text{s.t.} \quad & F_w + y_w^+ - y_w^- \ge F_v \\
& \quad + y_v^+ - y_v^- + t_w + u_w\cdot\xi_w && \forall\,(v,w)\in A \\
& y_{v_0}^+ = y_{v_0}^- = 0 \\
& y_v^+, y_v^- \ge 0 && \forall\, v\in V
\end{aligned} \tag{4}$$

In order to get robustness against all possible scenarios, we need to take the maximum deviation costs for all vectors $\xi\in\mathcal{U}_\Gamma$ representing scenarios. This leads to the problem

$$\max_{\xi\in\mathcal{U}_\Gamma} f(\xi,F) \tag{5}$$

We will refer to this problem as the second-stage problem. Moreover, we want to minimize the total costs associated with the initial schedule, so that we can describe the whole two-stage problem in the following way:

$$\min_{F\ge 0}\left\{\sum_{v\in V} c_v\cdot F_v + \max_{\xi\in\mathcal{U}_\Gamma} f(\xi,F)\right\}$$

By moving the maximization to the constraints we can recast the problem in the equivalent form:

$$\min_{F\geq 0,\Omega} \sum_{v\in V} c_v F_v + \Omega$$
$$\text{s.t. } \Omega \geq f(\xi, F) \qquad \forall \xi \in \mathcal{U}_\Gamma \tag{6}$$

In the following sections, we will analyze the second stage problem in more detail.

### 4.1. The second stage problem

We start this section by showing strong NP-hardness of the second stage problem (5).

**Theorem 1.** *The second stage problem (5) is strongly NP-hard.*

**Proof.** This reduction is inspired by Holzhauser et al. (2016, Theorem 9). We reduce from the ExactCoverBy3Sets-problem [X3C], which is known to be strongly NP-complete (Garey and Johnson, 1979, Problem SP2):

INSTANCE: Set $M = \{m_1, \dots, m_{3q}\}$ with $3q$ elements and a collection $C = \{C_1, \dots, C_k\}$ of 3-element subsets of $M$.
QUESTION: Does there exist a subcollection $C' \subseteq C$ such that every element $m_j \in M$ is contained in exactly one of the subsets $C'$?

The reduction is visualized in Fig. 1. We define $n = 3q + k + 1$ and tasks $V = \{v_0, v_{n+1}, s, v_1, \dots, v_k, v'_1, \dots, v'_{3q}\}$ with predecessor relationships denoted by the following arcs:

$$A = \{(s, v_i) : i = 1, \dots, k\} \cup \{(v_i, v'_j) : i = 1, \dots, k \wedge j = 1, \dots, 3q \wedge x_j \in C_i\}$$

The vertices $v_0$ and $v_{n+1}$ are dummy tasks needed by the formulation. Each task, except the dummy tasks, has processing time $t_v = 1$. The worst-case delay is given by

$$u_v = \begin{cases} 1 & \text{if } v = v_i \text{ for } i = 1, \dots, k \\ 0 & \text{else} \end{cases}$$

and we set $\Gamma = q$. Thus, only tasks corresponding to a set from $C$ can be delayed by up to one time step. The first-stage schedule is given by

$$F_v = \begin{cases} 0 & \text{if } v \in \{s, v_0\} \\ 1 & \text{if } v \in \{v_1, \dots, v_k\} \\ 2 & \text{if } v \in \{v'_1, \dots, v'_{3q}\} \\ 3 & \text{if } v \in \{v_{n+1}\} \end{cases}$$

If there are no delays, the schedule is feasible and each task starts exactly when all its predecessor tasks finish. Thus, $F_v - F_w + t_w = 0$ for each arc $(v, w) \in A$ and each delay of a task will also cause the same delay in all its successor tasks. The same holds for tasks started earlier than scheduled by $F$. Finally, we define the costs per time unit for finishing task $v \in V$ earlier or later than in the schedule as

$$d_v^+ = \begin{cases} 1 & \text{if } v \in \{v'_1, \dots, v'_{3q}\} \\ 0 & \text{else} \end{cases} \qquad d_v^- = \begin{cases} 3q & \text{if } v = s \\ 0 & \text{else} \end{cases}$$

Delaying a task $v$ has only costs for vertices of type $v'_j$. However, since there are no buffer times in the schedule $F$ and each task $v \in V$ will eventually be followed by a task $v'_j$, any delay will lead to a delay in a task $v'_j$. Similarly, when starting any task early, task $s$ has to start early as well.

The schedule $F$ is defined such that the leftmost task $s$ runs in the time frame $[0, 1]$. Then, all tasks of the second column $v_i$ run in the time frame $[1, 2]$ and finally, all tasks in the third column run in the time frame $[2, 3]$. The gray vertices are the dummy tasks.

Above each vertex, the costs for delay or early start are given in the form $[d_v^+, d_v^-]$. The blue vertices represent the only tasks that can be delayed. If such a task is delayed, we have to start the task early or delay all successor tasks. Due to the high costs of starting $s$ earlier, delaying all successor tasks will always be the better option.

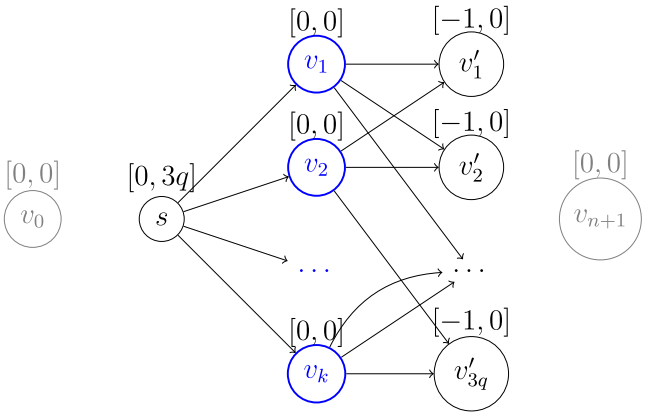



**Fig. 1.** Visualization $G$ for the reduction.

When delaying all successors, the worst case would consist of having to delay all tasks $v'_j$ for $j \in \{1, \dots, 3q\}$, which results in costs of $3q$. In this case, each vertex $v'_j$ must have a predecessor $v_i$ delayed by $1 = \xi^+_{v_i}$. This delayed predecessor vertex corresponds to an $m_j \in C_i$. With $\sum_{v\in V} |\xi_v| \leq \Gamma = q$, there can be at most $q$ such delayed $v_i$.

Thus, in this case, a collection of all sets $C_i \in C$ whose corresponding vertices are delayed solves the instance of the X3C-instance. Finally, we proof the following proposition:

There is a solution for the instance of the ExactCoverBy3Set-Problem if and only if the optimal solution to the second stage problem instance has objective value $\geq 3q$.

Let $C' \subseteq C$ with $|C'| = q$ be a solution to the instance of ExactCoverBy3Set. Then, we construct a solution of the second stage problem instance as follows:

$$\xi^+_w = \begin{cases} 1 & \text{if } w = v_i \text{ and } C_i \in C' \\ 0 & \text{else} \end{cases} \qquad \xi^-_w = 0$$

$$y^+_w = \begin{cases} 1 & \text{if } w = v'_j \\ 0 & \text{else} \end{cases} \qquad y^-_w = 0$$

The objective value is given by

$$\sum_{v\in V} d_v^- \cdot y_v^- + d_v^+ \cdot y_v^+ = \sum_{j=1}^{3q} d^+_{v'_j} \cdot y^+_{v'_j} = 3q$$

Including the delays $\xi$, we know that the task $s$ starts at time step 0 and finishes at time step 1, which is when all tasks $v_i$ start. They, in turn, finish one or two time steps later at 3 which is when all tasks $v'_j$ start if we include the delay $y^+$. Thus, the solution is feasible and has objective value $3q$. It remains to show that there is no better solution $\tilde{y}^+, \tilde{y}^-$ for the scenario $\xi$. Assuming that such a solution exists. Due to the structure of $G$ and the schedule $F$, it has to hold

$$F_s + \tilde{y}^+_s - \tilde{y}^-_s + t_s + t_{v_i} + u_{v_i} \leq F_{v_i} + \tilde{y}^+_{v_i} - \tilde{y}^-_{v_i} + t_{v_i} + u_{v_i} \leq F_{v'_j} + \tilde{y}^+_{v'_j} - \tilde{y}^-_{v'_j}$$

for each pair of $i, j$ with $m_j \in C_i \in C'$ and thus

$$\tilde{y}^+_s - \tilde{y}^-_s + 3 \leq 2 + \tilde{y}^+_{v'_j} - \tilde{y}^-_{v'_j} \quad \Leftrightarrow \quad 1 \leq \left(\tilde{y}^+_{v'_j} - \tilde{y}^-_{v'_j}\right) - \left(\tilde{y}^+_s - \tilde{y}^-_s\right).$$

Since $C'$ is a solution to X3C, there is a $C_i \in C'$ with $m_j \in C_i$ for each $m_j \in M$ and the inequality has to hold for each $j \in \{1, \dots, 3q\}$. Let $\delta \in [0, 1]$ with $\delta = \tilde{y}^-_s - \tilde{y}^+_s \leq \tilde{y}^-_s$ the time that task $s$ is started early. Then, $1 - \delta \leq \tilde{y}^+_{v'_j} - \tilde{y}^-_{v'_j} \leq \tilde{y}^+_{v'_j}$ equals the time that each task $v'_j$ has to be

delayed. With this, we get a lower bound on the objective value as

$$\sum_{v\in V} d_v^- \cdot \tilde{y}_v^- + d_v^+ \cdot \tilde{y}_v^+ = 3q \cdot \tilde{y}_s^+ + \sum_{j=1}^{3q} 1 \cdot \tilde{y}_{v'_j}^+ \geq 3q \cdot \delta + \sum_{j=1}^{3q} 1 \cdot \tilde{y}_{v'_j}^+ (1-\delta)$$
$$= 3q\delta + (1-\delta)3q = 3q$$

Thus, $y$ is already optimal.

Now, let $\xi, y$ be an optimal solution to the second stage problem with objective value $3q$. Let

$$\tilde{y}_w^+ = \begin{cases} \max\limits_{i\,:\,m_j \in C_i} \xi_{v_i}^+ & \text{if } w = V'_j \\ 0 & \text{else} \end{cases} \qquad \tilde{y}_w^- = 0$$

be an alternate solution to $y$. It is easy to see that $y$ is feasible since each vertex $v'_j$ is delayed as much as necessary for feasibility by definition. Since $y$ is optimal with objective value $\geq 3q$, we know that the objective value of $\tilde{y}$ is at least $3q$. The objective value of $\tilde{y}$ is given by

$$3q \leq \sum_{v\in V} d_v^- \cdot \tilde{y}_v^- + d_v^+ \cdot \tilde{y}_v^+ = \sum_{j=1}^{3q} 1 \cdot \overbrace{\tilde{y}_{v'_j}^+}^{\leq 1} \Rightarrow \tilde{y}_{v'_j}^+ = 1 \text{ for each } j \in \{1,\dots,3q\}$$

However, this can only be the case if there is a $C_i$ with $\xi_{v_i}^+ = 1$ and $m_j \in C_i$ for each $j \in \{1,\dots,3q\}$. With $\Gamma = 1$, there can be at most $q$ such sets $C_i$, thus we can define a collection $C' = \{C_i : \xi_{v_i}^+ = 1\}$ such that each $m_j \in M$ is in exactly one set of $C'$. This collection then solves the instance of the X3C problem.

This concludes the proof of correctness and thus the reduction.

By the result of Theorem 1 it is unlikely to find an efficient algorithm to solve the second stage problem. In order to solve the problem anyway, we will now introduce a compact, nonlinear formulation for the second stage problem.

**Theorem 2.** *The second stage problem (5) is equivalent to the following compact, nonlinear program*

$$\begin{aligned} \max_{x,\xi^+,\xi^-} \ & \sum_{a=(v,w)\in A} (F_v - F_w + t_w + \xi_w^+ \cdot u_w - \xi_w^- \cdot u_w) \cdot x_a \\ s.t. \ & \sum_{a=(\cdot,v)\in A} x_a - \sum_{a=(v,\cdot)\in A} x_a \leq d_v^+ && \forall v \in V\backslash\{v_0\} \\ & -\sum_{a=(\cdot,v)\in A} x_a + \sum_{a=(v,\cdot)\in A} x_a \leq d_v^- && \forall v \in V\backslash\{v_0\} \\ & x_a \geq 0 && \forall a \in A \\ & \sum_{v\in V} \xi_v^+ + \xi_v^- \leq \Gamma \\ & 0 \leq \xi_v^+, \xi_v^- \leq 1 && \forall v \in V. \quad (7) \end{aligned}$$

**Proof.** In order to get a compact formulation for the second stage problem, we first dualize the linear Program (4) for fixed $\xi$ and $F$. We get the following problem:

$$\begin{aligned} f(\xi,F) = \max_{x} \ & \sum_{a=(v,w)\in A} (F_v - F_w + t_w + \xi_w \cdot u_w) \cdot x_a \\ \text{s.t.} \ & \sum_{a=(\cdot,v)\in A} x_a - \sum_{a=(v,\cdot)\in A} x_a \leq d_v^+ && \forall v \in V\backslash\{v_0\} \\ & -\sum_{a=(\cdot,v)\in A} x_a + \sum_{a=(v,\cdot)\in A} x_a \leq d_v^- && \forall v \in V\backslash\{v_0\} \\ & x_a \geq 0 && \forall a \in A \end{aligned}$$

First, we will shortly discuss an interpretation of this problem.

For each arc $a = (v,w) \in A$, the objective of $x_a$ represents the costs caused by the (possibly negative) deviation from the schedule for task $w$ under the assumption that it can be started at the finish time of task $v$. More precisely, $F_w - F_v$ is the scheduled duration of the task under this assumption and $t_w + \xi_w \cdot u_w$ the realized duration. If this value is positive, the task is delayed by taking longer than anticipated or because of a previous delay. If it is negative, the task is earlier or there is some buffer time between tasks $v$ and $w$. Accordingly, $x_{(v,w)}$ represents the cost per time unit caused by the delay of task $w$ on the whole graph. This includes not only the costs caused by $w$ itself, but also the costs for the delay of later tasks depending on $w$.

Next, we insert this formulation for $f(\xi,F)$ into the second stage problem $\max_{\xi\in\mathcal{U}_\Gamma} f(\xi,F)$. This is now a quadratic problem since we optimize over both $\xi$ and $x$. By substituting $\xi$ for $\xi^+ - \xi^-$ to deal with the inequality constraint $\sum_{v\in V} |\xi_v| \leq \Gamma$, we can formulate this problem as

$$\begin{aligned} \max_{x,\xi^+,\xi^-} \ & \sum_{a=(v,w)\in A} (F_v - F_w + t_w + \xi_w^+ \cdot u_w - \xi_w^- \cdot u_w) \cdot x_a \\ \text{s.t.} \ & \sum_{a=(\cdot,v)\in A} x_a - \sum_{a=(v,\cdot)\in A} x_a \leq d_v^+ && \forall v \in V\backslash\{v_0\} \\ & -\sum_{a=(\cdot,v)\in A} x_a + \sum_{a=(v,\cdot)\in A} x_a \leq d_v^- && \forall v \in V\backslash\{v_0\} \\ & x_a \geq 0 && \forall a \in A \\ & \sum_{v\in V} \xi_v^+ + \xi_v^- \leq \Gamma \\ & 0 \leq \xi_v^+, \xi_v^- \leq 1 && \forall v \in V. \end{aligned}$$

This concludes the proof.

We now have a compact formulation of the second stage problem. However, the formulation is nonlinear and the problem is not convex, thus we cannot solve it using the polynomial time algorithm by Ye and Tse (1989). Because of this, we will now introduce a linear formulation that is not compact and utilizes a cutting plane type algorithm similar to the approach by Terry et al. (2009).

Let $X$ be the set of feasible dual variables

$$X := \left\{ x \in \mathbb{R}_{\geq 0}^{|A|} : -d_v^- \leq \sum_{a=(\cdot,v)\in A} x_a - \sum_{a=(v,\cdot)\in A} x_a \leq d_v^+ \ \forall v \in V\backslash\{v_0\} \right\}$$

and

$$\tilde{\mathcal{U}}_\Gamma = \left\{ (\xi^+,\xi^-) \in (\mathbb{R}_{\geq 0}^{|V|})^2 : \xi^+ - \xi^- \in \mathcal{U}_\Gamma \right\}$$

the new uncertainty set. We assume that there is an enumeration of the extreme points of $X$, i.e., a set $\tilde{X} = \{x^i : 1 \leq i \leq K\} \subseteq \mathbb{R}_{\geq 0}^{|A|}$ for some $K \in \mathbb{N}$. For $\xi$ and $F$ fixed, the linear program $f(\xi,F)$ is convex and therefore it takes its optimal value at an extreme point $x^i \in \tilde{X}$ and we can rewrite the second stage problem(s) as

$$\max_{1\leq k\leq K} \Phi(F,t,x^k,u)$$

with

$$\Phi(F,t,x^k,u) := \sum_{a=(v,w)\in A} (F_v - F_w + t_w) x_a^k + \max_{(\xi^+,\xi^+-)\in\tilde{\mathcal{U}}_\Gamma} (\xi_w^+ \cdot u_w - \xi_w^- \cdot u_w) x_a^k.$$

Note, that

$$\max_{1\leq k\leq K} \Phi(F,t,x^k,u) = \max_{\xi\in\mathcal{U}_\Gamma} f(\xi,F)$$

since both formulations of the second stage problem are equivalent. Thus, we can now insert this formulation into the complete problem analogous to (6).

$$\begin{aligned} \min_{F\geq 0,\Omega} \ & \sum_{v\in V} c_v F_v + \Omega \\ \text{s.t.} \ & \Omega \geq \Phi(F,t,x^k,u) && \forall 1 \leq k \leq K \quad (8) \end{aligned}$$

Note, that for each $k \in K$ there is an inner maximization problem of finding the worst-case scenario in $\Phi$. For fixed $x^k$, we can now dualize the inner problem

$$\max_{(\xi^+,\xi+^-)\in\tilde{\mathcal{U}}_\Gamma} (\xi_w^+ \cdot u_w - \xi_w^- \cdot u_w) x_a^k$$

and get the following equivalent minimization problem:

$$\min_{\theta^k,\lambda^{k,+},\lambda^{k,-}} \Gamma\theta^k + \sum_{v\in V} (\lambda_v^{k,+} + \lambda_v^{k,-})$$

$$\text{s.t. } \theta^k + \lambda_v^{k,+} \geq \sum_{w\in V\,:\,(w,v)\in A} u_v \cdot x^k_{(w,v)} \qquad \forall v \in V$$
$$\theta^k + \lambda_v^{k,-} \geq -\sum_{w\in V\,:\,(w,v)\in A} u_v \cdot x^k_{(w,v)} \qquad \forall v \in V$$
$$\theta^k, \lambda^{k,+}, \lambda^{k,-} \geq 0$$

Now, both the inner and the outer problem are minimization problems and we can shift the minimization from the constraint to the objective. The whole problem can then be rewritten as

$$\min_{F,\Omega,\theta,\lambda^+,\lambda^-} \sum_{v\in V} c_v F_v + \Omega \tag{9}$$
$$\text{s.t. } \Omega \geq \sum_{\substack{a\in A\\ a=(v,w)}} (F_v - F_w + t_w)x^k_a + \Gamma\theta^k + \sum_{v\in V}(\lambda_v^{k,+}+\lambda_v^{k,-}) \qquad \forall 1 \leq k \leq K$$
$$\theta^k + \lambda_v^{k,+} \geq \sum_{w\in V\,:\,(w,v)\in A} u_v \cdot x^k_{(w,v)} \qquad \forall v \in V,\ 1 \leq k \leq K$$
$$\theta^k + \lambda_v^{k,-} \geq -\sum_{w\in V\,:\,(w,v)\in A} u_v \cdot x^k_{(w,v)} \qquad \forall v \in V,\ 1 \leq k \leq K$$
$$F, \theta^k, \lambda^{k,+}, \lambda^{k,-} \geq 0 \qquad \forall 1 \leq k \leq K$$

We now have an LP formulation of the problem which can be used to solve the problem optimally. However, we cannot efficiently calculate all vertices of $\tilde{X}$ and $K$ can be exponential in $|V|$. Therefore, computing a schedule by solving this LP directly is very inefficient. Instead, we now present an algorithmic approach based on separation.

### 4.2. Exact algorithm based on vertex enumeration

In order to deal with the exponentially many constraints in the LP formulation (9) we use a cutting plane type algorithm. Therefore, for a given solution $F^*, \Omega^*$, we need to find a violated constraint. We will use the equivalent formulation (8) because it has only one kind of constraint and finding a violated constraint is equivalent to finding a vertex $x \in \tilde{X}$ with $\Omega^* < \Phi(F^*, t, x^k, u)$. In order to find this constraint or prove optimality, we need to compute the vertex $x \in \tilde{X}$ that maximizes the right-hand side of the inequality and compare it with $\Omega$. This is equivalent to computing

$$\max_{1\leq k\leq K} \Phi(F^*, t, x^k, u)$$

which is the compact formulation of the subproblem (7). Therefore, the separation problem can be reduced to finding an optimal solution of the subproblem and we again face the same non-convex quadratic program we considered previously. The only difference is that we now can assume $F = F^*$ to be fixed. In order to remove the quadratic term, we will rewrite the program using McCormick envelopes, a technique that is for example presented in Tsoukalas and Mitsos (2014). Usually, McCormick envelopes are used as a relaxation technique and we will use this for our first heuristic. However, we will not use a relaxation here but an exact reformulation. This is possible, since the set $\tilde{U}_\Gamma$ defines a polyhedron. For fixed $x$, the maximization problem is linear in $\xi^+, \xi^- \in \tilde{U}_\Gamma$ and each solution of this problem can be defined by some extreme point with $\xi^+, \xi^- \in \{0,1\}^{|V|}$ as the matrix of constraints is totally unimodular. With this assumption, we can linearize the subproblem by introducing new variables $q^+_{(v,w)}$ and $q^-_{(v,w)}$ with $q^+_{(v,w)} = \xi^+_w \cdot x_{(v,w)}$ and $q^-_{(v,w)} = \xi^-_w \cdot x_{(v,w)}$ for all $(v,w) \in A$. We can assure these inequalities by adding the following constraints using a constant $M > 0$ sufficiently large:

$$0 \leq q^+_a, q^-_a \leq x_a \qquad \forall a \in A$$
$$M \cdot \xi^+_w + x_{(v,w)} - M \leq q^+_{(v,w)} \leq M \cdot \xi^+_w \qquad \forall (v,w) \in A$$
$$M \cdot \xi^-_w + x_{(v,w)} - M \leq q^-_{(v,w)} \leq M \cdot \xi^-_w \qquad \forall (v,w) \in A$$

It is easy to see that for $\xi^+, \xi^- \in \{0,1\}^{|V|}$ these inequalities assure the desired equalities. Thus, by adding these constraints and replacing the quadratic term in the objective we get the following mixed integer program

$$\max_{x,\xi^+,\xi^-} \sum_{a=(v,w)\in A} (F_v - F_w + t_w)x_a + q^+_a \cdot u_w - q^-_a \cdot u_w$$
$$\text{s.t. } \sum_{a=(\cdot,v)\in A} x_a - \sum_{a=(v,\cdot)\in A} x_a \leq d^+_v \qquad \forall v \in V\backslash\{v_0\}$$
$$-\sum_{a=(\cdot,v)\in A} x_a + \sum_{a=(v,\cdot)\in A} x_a \leq d^-_v \qquad \forall v \in V\backslash\{v_0\}$$
$$x_a \geq 0 \qquad \forall a \in A$$
$$\sum_{v\in V} \xi^+_v + \xi^-_v \leq \Gamma$$
$$q^+_a, q^-_a \leq x_a \qquad \forall a \in A$$
$$q^+_{(v,w)} \leq M \cdot \xi^+_w \qquad \forall (v,w) \in A$$
$$q^-_{(v,w)} \leq M \cdot \xi^-_w \qquad \forall (v,w) \in A$$
$$q^+_{(v,w)} \geq M \cdot \xi^+_w + x_{(v,w)} - M \qquad \forall (v,w) \in A$$
$$q^-_{(v,w)} \geq M \cdot \xi^-_w + x_{(v,w)} - M \qquad \forall (v,w) \in A$$
$$x_a, q^+_a, q^-_a \geq 0 \qquad \forall a \in A$$
$$\xi^+_v, \xi^-_v \in \{0,1\} \qquad \forall v \in V \tag{10}$$

There are two import things to note concerning this formulation. First, it is a mixed integer problem and therefore it is not clear weather it can be solved in polynomial time. Second, this program is only linear for fixed $F$ and this technique cannot be used to solve the complete problem exactly. It can, however, be used for a heuristic, as we will see in the next chapter.

This formulation can now be used to solve the problem exact with a cutting plane algorithm. Starting with a (possibly empty) subset $\hat{X} \subseteq \tilde{X}$ in each iteration, the LP (9) with only constraints corresponding to the nodes from $\hat{X}$ is solved optimally. For the optimal solution $F^*, \Omega^*$, a new vertex $x^*$ is computed by solving (10). If the objective is greater than $\Omega^*$, a violated constraint is found and we add $x^*$ to $\hat{X}$. Else, the solution $F^*, \Omega^*$ is optimal.

**Algorithm 1** Exact algorithm for the $\Gamma$-robust two-stage project scheduling problem

**Input:** Graph $G = (V, A)$, uncertainty parameter $\Gamma \in \mathbb{N}_{\leq|V|}$, execution times $(t_v)_{v\in V}$, uncertainties $(u_v)_{v\in V}$, costs per time unit $(c_v)_{v\in V}$, costs per time unit for finishing earlier than scheduled $(d^-_v)_{v\in V}$, costs per time unit for finishing later than scheduled $(d^+_v)_{v\in V}$, initial vertex set $\hat{X} \subset \tilde{X}$, sufficiently large constant $M$

**Output:** the costs for the $\Gamma$-robust two-stage problem and the finish times for all tasks $(F_v)_{v\in V}$

```
while true do
    schedule_obj, F*, Ω* optimal solution of (9) with constraints only for X̂
    subproblem_obj, x* = optimal solution of (10)
    if subproblem_obj > Ω then
        X̃ = X̃ ∪ {x}
    else
        break;
    end if
end while
return schedule_obj, (F_v)_{v∈V}
```

Algorithm 1 finds an optimal solution for LP (9) and therefore solves the $\Gamma$-robust two-stage PSP optimally. Since in each iteration a MIP has to be solved, the runtime is possibly exponential. Therefore, in practice it might be better to not finish the algorithm, but stop early and only get an approximation on the optimal value. One possibility is to stop the computation after a certain time limit. This is useful when a solution needs to be found fast. Another possibility is to use the bounds computed in each iteration. The solution of the (to $\hat{X}$ restricted)

problem *schedule_obj* poses a lower bound on the optimal solution, since it does not fulfill all constraints and is therefore better than the optimal solution. On the other hand, the objective of the subproblem *subproblem_obj* can be used to compute an upper bound, since $F^*$ and $\Omega' = subprblem\_obj$ are a feasible solution of (8) and therefore the objective value $\sum_{v\in V} F_v^* c_v + \Omega'$ is an upper bound to the minimization problem. For an optimal solution holds $\Omega^* = \Omega'$ and therefore lower and upper bound are equal. Thus, the gap between these bounds can be used as a quality measure of the solution and we can stop the computation if it is lower than a certain value $\varepsilon$.

We now have an iterative algorithm which computes an optimal solution and can be naturally used as a heuristic by constricting the number of iterations. In the next subsections, we will introduce two more heuristics to get good, though not necessarily optimal, solutions faster.

### 4.3. McCormick heuristic

As mentioned before, we will now use McCormick envelopes in order to approximate the optimal value. To this end, we will use the MIP formulation of the subproblem (10). It should be noted that, even though in (7) we could assume $\xi^+, \xi^- \in \{0,1\}^{|V|}$ for any optimal solution, this does not hold in the linear relaxation of the MIP formulation. Thus, relaxing the constraint $\xi^+, \xi^- \in \{0,1\}^{|V|}$ to $\xi^+, \xi^- \in [0,1]^{|V|}$ does not yield an optimal solution but a lower bound. Since the linear relaxation of the MIP for fixed $F$ is a linear program, we can transform it into a minimization problem by dualizing it. That way, a compact formulation for the whole problem can be created by merging the minimization over $F$ and the minimization of the subproblem. It is then given by

$$
\begin{aligned}
\min_{\substack{F,\mu^\pm,\theta,\alpha^\pm\\ \beta^\pm,\lambda^\pm,\phi^\pm}} & \sum_{v\in V\setminus\{v_0\}} \left(d_v^-\mu_v^- + d_v^+\mu_v^+\right) + \sum_{v\in V}\left(c_v\cdot F_v + \phi_v^+ + \phi_v^-\right) \\
& + \Gamma\theta + M\cdot\sum_{a\in A}\left(\lambda_a^+ + \lambda_a^-\right) \\
\text{s.t. } & \mu_w^+ - \mu_w^- - \alpha_a^+ - \alpha_a^- + \lambda_a^+ + \lambda_a^- \geq -F_w + t_w && \forall a=(v_0,w)\in A \\
& \mu_w^+ - \mu_v^+ - \mu_w^- + \mu_v^- - \alpha_a^+ - \alpha_a^- + \lambda_a^+ + \lambda_a^- && \forall a=(v,w)\in A \\
& \quad \geq F_v - F_w + t_w && v\neq v_0 \\
& \theta - M\cdot\sum_{a=(\cdot,v)\in A}(\beta_a^+ - \lambda_a^+) + \phi_v^+ \geq 0 && \forall v\in V \\
& \theta - M\cdot\sum_{a=(\cdot,v)\in A}(\beta_a^- - \lambda_a^-) + \phi_v^- \geq 0 && \forall v\in V \\
& \alpha_a^+ + \beta_a^+ - \lambda_a^+ \geq u_w && \forall\, a=(v,w)\in A \\
& \alpha_a^- + \beta_a^- - \lambda_a^- \geq -u_w && \forall\, a=(v,w)\in A \\
& F,\mu^+,\mu^-,\theta,\alpha^+,\alpha^-,\beta^+,\beta^-,\lambda^+,\lambda^-,\phi^+,\phi^-\geq 0 && (11)
\end{aligned}
$$

Now, we have a compact formulation as a linear program that we can solve efficiently and which produces a baseline schedule.

In order to calculate the real costs for the schedule, we apply the subproblem to the given baseline schedule. Since we want to avoid the non-convex quadratic problem, we will use the rewritten version (10) used to determine the next vertex in the exact algorithm. We also need to keep in mind that this formulation is based on a relaxation. The quality of the solution highly depends on the choice of the parameter $M$.

- In case the value of $M$ is chosen sufficiently large, the deviation costs might be overestimated. For example, for $M = 2x_{(v,w)}$ and $\xi_w^+ = 0.5$, we get $0 \leq q_{(v,w)}^+ \leq x_{(v,w)}$ as single constraint on $q_{(v,w)}^+$ and thus $q_{(v,w)}^+ = x_{(v,w)}$ is feasible even though $\xi_w^+ = 0.5$ and, in the exact formulation, $q_{(v,w)}^+ \equiv x_{(v,w)}\cdot\xi_w^+$. Thus, for large values of $M$ the effective scenario $q_{(v,w)}^+/x_{(v,w)}$ and therefore $\Gamma$ can be much higher than intended.
- In case the value of $M$ is chosen too small, only values with $x_a \leq M$ lead to feasible solutions. Hence, the feasible set of $x$ is restricted and each delay can only cause costs of at most $M$, so that the deviation costs in total might be underestimated and the estimated costs might not equal the actual robust costs for the produced schedule.

To learn more about how $M$ affects the objective value, see Appendix. In summary, the complete heuristic based on McCormick consists of first computing a baseline schedule $F^*$ by solving (11) optimally for a fixed $M$ and then calculation its cost by solving (10) for $F^*$. We can easily see that

$$x_a \leq \sum_{v\in V} d_v^+ =: \tilde{M}$$

must hold. Thus, we only need to consider $0 \leq M \leq \tilde{M}$. Now, we could choose multiple (or just one) fixed values for $M$ from this interval and use them as a parameter for the McCormick heuristic, finally picking the best result. We call these approaches *static* since $M$ is chosen fixed and independent of the problem instance.

Alternatively, we could also choose the parameter $M$ adaptively by using a (steepest) descent method to find the $M$ that minimizes the objective value of the solution computed by the McCormick heuristic, see Algorithm 2. We start with $M = \tilde{M}$. At each iteration, we take a step to a new value of $M$ and check, whether the new result of the McCormick heuristic with that $M$ is better than the previous one. If the solution is better, we continue with that step-width and direction. If not, we change direction and decrease the step-width. We call this heuristic the Adaptive McCormick heuristic.

**Algorithm 2** Adaptive McCormick heuristic for the $\Gamma$-robust two-stage project scheduling problem

**Input:** Graph $G=(V,A)$, uncertainty parameter $\Gamma\in\mathbb{N}_{\leq|V|}$, execution times $(t_v)_{v\in V}$, uncertainties $(u_v)_{v\in V}$, costs per time unit $(c_v)_{v\in V}$, costs per time unit for finishing earlier than scheduled $(d_v^-)_{v\in V}$, costs per time unit for finishing later than scheduled $(d_v^+)_{v\in V}$

**Output:** the costs for the $\Gamma$-robust two-stage problem and the finish times for all tasks $(F_v)_{v\in V}$

```
step = -1/2
M_opt = M̃
cost_opt, F_opt optimal solution of McCormick heuristic with M = M_opt
while step ≥ 0.01 and number iterations ≤ 30 do
    M_temp = M_opt · (1 + step)
    cost_temp, F_temp optimal solution of McCormick heuristic with M = M_temp
    if cost_temp < cost_opt then
        M_opt, cost_opt, F_opt = M_temp, cost_temp, F_temp
    else
        step = -1/2 · step
    end if
end while
return cost_opt, F_opt
```

As mentioned above, the objective value of (11) is only an approximation on the robust deviation costs. In order to compute the actual robust deviation costs for a produced baseline schedule, a mixed integer program (10) is used. Thus, the algorithm is not necessarily polynomial time. However, computational results show that it can be solved very fast. In order to get an efficient heuristic, the step of recalculating the costs could be skipped. For sufficiently large constants $M$, the costs of the returned schedule might then be overestimated and provide an upper bound on the robust costs for the baseline schedule. That way, the heuristic is efficient, but the calculated costs are overestimated. For values of $M$ that are too small, the costs of the produced schedule might be underestimated, so that they can be exceeded in practice. Thus, the mixed integer problem should be included in order to get accurate results.

### 4.4. Single-stage heuristic

In Section 3, we derived an algorithm for the single stage problem, which determines the earliest guaranteed finish times for all task. Thus, using the resulting schedule for the two-stage problem will lead to no additional costs for late completion. Additionally, no costs for early completion need to be paid since the project planner can simply decide to wait for the next task in case a task is finished earlier instead of paying the early deviation penalty. The only necessary adjustment is returning the cost of the $\Gamma$-robust schedule instead of only returning the schedule itself. Thus, the single-stage heuristic consist of computing an optimal solution $F^*$ for the single-stage formulation using the recursive algorithm defined by (3) and returning that solution together with its cost $c^T F^*$. Note, that here we do not need to compute the solution of the subproblem in order to compute the costs as its optimal value will always be 0 in this case.

As seen in Section 3, this algorithm can be computed in $\mathcal{O}(|V|^3)$ since we did not change the computation of the $F_{v,\gamma}$. It leads to an upper bound on the costs since it leads to a feasible solution for the two-stage problem, but not necessarily a minimal one.

This algorithm will always compute a very conservative schedule that is feasible for each scenario. To include more flexibility in the solution, we could use the single-stage formulation to compute an optimal solution for a smaller value of $\tilde{\Gamma} < \Gamma$ and then use the resulting schedule as the baseline schedule. Note that in this case we must compute the solution of the subproblem, since this schedule is not robust against any scenario with $\Gamma$ delays. Despite the subproblem being NP-hard, computational experiments show that we can solve it relatively fast. Thus, we propose the adaptive single-stage heuristic, in which we compute the optimal schedule for the single-stage problem for each $\tilde{\Gamma} \in \{0, \dots, \Gamma\}$ and use it as a baseline schedule. For each of these schedules, we compute its objective value as a baseline schedule in the two-stage problem with $\Gamma$ and choose the best schedule. This algorithm will compute less conservative schedules, resulting in higher costs in the second stage, but hopefully lower costs overall. Applying this heuristic approach to the resource-constrained PSP results in the two-stage adjustable robust RCPSP proposed by Bruni et al. (2017).

### 4.5. Iterative recovery heuristic

In the second-stage problem introduced in Section 4.1, for a given baseline schedule $F$, a worst-case scenario $\xi$ and corrections $y$ are computed for the baseline schedule.

Hence, we propose an iterative heuristic that, starting from the deterministic solution ignoring all possible delays, adapts the baseline schedule by one step in the direction indicated by the corrections $y$. Again, computing the exact solution for the second stage problem is NP-hard but relatively fast in practice using the MIP formulation. The complete algorithm is then given by Algorithm 3.

We chose to terminate if an iteration step does not lead to a (relative) improvement of the objective value of at least 0.001%. This condition can be relaxed, however we would need to ensure that the algorithm does not run in a cycle. In addition, we chose an upper bound of 30 iterations to limit the computation time of the heuristic. Computational experiments showed that this bound gives the best trade-off between computation time and solution quality. It was rarely met and the computation took less than 12 s on average for all instances tested. Finally, we would like to mention the possibility of computing the second stage solution through a heuristic. However, due to the speed of current solvers on these problems, the small gain in computation time is not worth the loss in solution quality.

**Algorithm 3** Iterative Recovery Heuristic

**Input:** Graph $G = (V, A)$, uncertainty parameter $\Gamma \in \mathbb{N}_{\leq |V|}$, execution times $(t_v)_{v \in V}$, uncertainties $(u_v)_{v \in V}$, costs per time unit $(c_v)_{v \in V}$, costs per time unit for finishing earlier than scheduled $(d_v^-)_{v \in V}$, costs per time unit for finishing later than scheduled $(d_v^+)_{v \in V}$

**Output:** the costs for the $\Gamma$-robust two-stage problem and the finish times for all tasks $(F_v)_{v \in V}$

```
F solution of the deterministic project scheduling problem G, t
ξ, y optimal solution of the second stage problem for schedule F with cost obj
while number iterations ≤ 30 do
    F̃ with F̃_v = F_v + 1 if y_v > 0 and F̃_v = F_v − 1 if y_v < 0
    ξ̃, ỹ optimal solution of the second stage problem for schedule F̃ with cost õbj
    if õbj/obj > 1 − 10^−5 then
        F, ξ, y, obj = F̃, ξ̃, ỹ, õbj
    else
        break
    end if
end while
return obj, F
```

## 5. Computational study

Based on the algorithms presented in the previous sections, we conduct a computational study. First, we analyze the convergence of the enumeration algorithm 1 on instances of different sizes. Next, we compare the McCormick heuristic introduced in Section 4.3 for three different values of $M$, the adaptive McCormick heuristic, the single-stage heuristic, the adaptive single-stage heuristic and the iterative recovery heuristic on small instances. Lastly, we compare the results of the enumeration algorithm with restricted runtime to the best heuristics on larger instances.

All approaches were programmed in Python and tested on the benchmark problems for the deterministic recourse constrained PSP obtained from the online library PSPLIB (Kolisch and Sprecher, 1997). All models were coded in Python and solved using Gurobi 9.1.2, running on an Intel Core i5-3570 CPU 3.4 GHz RAM 16 GB.

### 5.1. Instances

For each instance, three parameters are given to measure its difficulty, two of which refer to the resource-constrained and are therefore of no value to us. The third parameter, called network complexity ($NC$), is given by the average number of non-redundant arcs per node. For each network complexity $NC \in \{1.5, 1.8, 2.1\}$ 160 instances are available. For each of the 480 instances, we consider three values $\{3, 5, 10\}$ for $\Gamma$. We randomly create the uncertainties $(u_v)_{v \in V}$, the costs $(c_v)_{v \in V}$, the penalty for finishing earlier than scheduled $(d_v^-)_{v \in V}$ and the penalty for finishing later than scheduled $\left(\tilde{d_v^+}\right)_{v \in V}$ as integer values within a specified range. Since finishing later than scheduled should be more expensive than using a late scheduling from the beginning, the costs $(c_v)_{v \in V}$ will be added to the late penalty $\left(\tilde{d_v^+}\right)_{v \in V}$, so that we get $d_v^+ := \tilde{d_v^+} + c_v$ for all $v \in V$. The ranges used to create these parameters can be found in Table 1. Since the nodes $v_0$ and $v_{n+1}$ are only artificial nodes that do not correspond to actual tasks, we will set $u_{v_0} := u_{v_{n+1}} := c_{v_0} := d_{v_0}^+ := d_{v_0}^- := d_{v_{n+1}}^- := 0$. The node $v_{n+1}$ defines the makespan of the project, so that we will still have a cost and a penalty for a delay of this node.

Finally, we need to consider the parameter $M$ used for both the exact algorithm and the McCormick heuristic. From the interpretation of the subproblem in Section 4.1, we know that

$$x_a \leq \sum_{v \in V} d_v^+ =: \tilde{M}.$$

Thus, we will use this value as a guaranteed upper bound for the exact algorithm. Since the results of the McCormick algorithm are more

**Table 1**
Bounds for the parameter settings.

| Parameter | Lower bound | Upper bound |
|---|---|---|
| $(u_v)_{v\in V\setminus\{v_0,v_{n+1}\}}$ | 0 | 10 |
| $(c_v)_{v\in V\setminus\{v_0\}}$ | 5 | 20 |
| $(d_v^-)_{v\in V\setminus\{v_0,v_{n+1}\}}$ | 5 | 10 |
| $(d_v^+)_{v\in V\setminus\{v_0\}}$ | 5 | 10 |

precise the closer the bound provided by $M$ is, we choose different values and compare their results. More precisely, we consider $a \in \{1, 0.5, 0.25\}$ and refer to the McCormick heuristic using this parameter as McCormick $a$. This leads to three versions of the McCormick heuristic with fixed parameter $M$ and thus seven heuristics and one exact algorithm in total.

### 5.2. Results

In this section, we first consider the runtime and convergence properties of the exact algorithm. Next, we present the overall performance of the heuristics for all instances with 30 tasks. Then, we will evaluate the influence of different values of $\Gamma$ on the heuristics. Last, we will compare the results of the heuristics to that of the enumeration algorithm with a runtime restriction of 10 min on larger instances.

#### 5.2.1. The enumeration algorithm

First, we want to analyze the convergence and runtime of the exact algorithm. For instances with 30 tasks, the algorithm finishes in under three minutes and computes an optimal solution for every instance. This is not the case for larger instances. First, we take a look at the convergence for one instance with 60 tasks shown in Fig. 2. The blue graph represents the value of the optimal solution $F^*, \Omega^*$ of the linear Program (9) with constraints only for $\hat{X}$ (starting with $\hat{X} = \emptyset$).

The red graph represents the value of that solution in the original problem, i.e., the optimal value $\Omega$ of the subproblem (10) plus the cost of the schedule $F^*$ given by $c^T F^*$. In other words, the red graph shows the value of a feasible solution and thus an upper bound on the optimal value and the blue graph is a feasible dual solution and a lower bound. In the following, we often refer to the normalized difference between the upper and lower bound as gap. The figure implies that the gap converges to zero but the program does not finish within the time limit of 10 min. Starting with a gap of over 500%, the gap falls to under 3% within in these first 10 min. For comparison, after 20 min, the gap

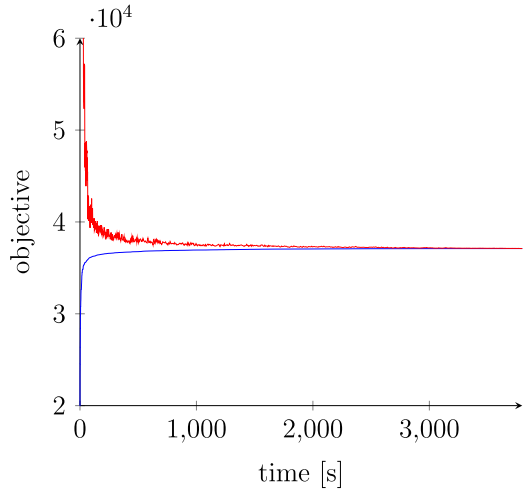



**Fig. 2.** Convergence of the exact enumerate algorithm for one instance with 60 tasks and 10 min.

**Table 2**
Deviation from the optimal value for the instances with 30 tasks.

| Heuristic | Mean [%] | SD [%] | Min [%] | Max [%] | Runtime [s] |
|---|---|---|---|---|---|
| McCormick 1 | 15.01 | 9.21 | 1.49 | 42.55 | 0.046 |
| McCormick 0.5 | 9.21 | 5.24 | 1.53 | 35.95 | 0.062 |
| McCormick 0.25 | 19.48 | 8.48 | 1.82 | 45.15 | 0.065 |
| Single-stage | 10.15 | 5.09 | 1.49 | 28.49 | 0.001 |
| Adaptive McCormick | 5.50 | 1.95 | 1.05 | 16.06 | 1.076 |
| Adaptive single-stage | 6.65 | 2.44 | 1.50 | 17.37 | 0.237 |
| Iterative recovery heuristic | 2.13 | 1.49 | 0.01 | 10.87 | 1.173 |

reduces to about 1% and the computation finished after 3800 s or about 60 min. As expected, most progress is made in the first few minutes of computation. Because of this, we restricted the computation time for the enumeration algorithm to 10 min. In this time frame, the smallest instances with 30 tasks can be solved to optimality while the bigger instances get solved with varying gaps.

Fig. 3 shows that this trend holds true for all instances. In this figure, the normalizes gap between lower bound and upper bound for all instances is displayed. Each of the four figures represents the results for a different number of tasks per instance. The blue graph represents the median over all instances with the corresponding number of tasks and the brown and red graph the minimal or maximal gap respectively. The green space is the interquartile range (IQR) in which the middle 50% of instances lie. To improve readability, the scale of the axes differ.

In all four cases, the gap first reduces very fast and then slowly converges toward zero. This behavior is more prevalent in the smaller instances with fewer tasks, where the gap first reduces more steeply than for the bigger instances. For instances with 30 or 60 tasks, the average gap is under 4% after 10 min and all instances with 30 tasks are solved to optimality within 140 s. For the bigger instances, the algorithm did not compute a good result in 10 min and would need to run longer in order to compute a good result.

Fig. 4 further illustrates the results after 10 min for the enumeration algorithm for different sized instances. It plots the cumulative percentage of instances solved to within a gap of optimality within ten minutes for instances with 30, 60, 90 and 120 tasks. The instances with 30 tasks were all solved to optimality. All instances with 60 tasks were solved with a gap of up to 13%, about 80% of them have a gap of 5% or less. Considering the instances with 90 tasks, 80% of them were solved with a gap of 25%. However, the algorithm computes a solution with gap >100% for two outliers. From the instances with 120 tasks, only 73.75% were solved with a gap of 100% or less. We also note that the curve for the instances with 120 tasks falls less steep than the other curves. This indicates, that the time limit of 10 min for the algorithm was chosen too low for larger instances.

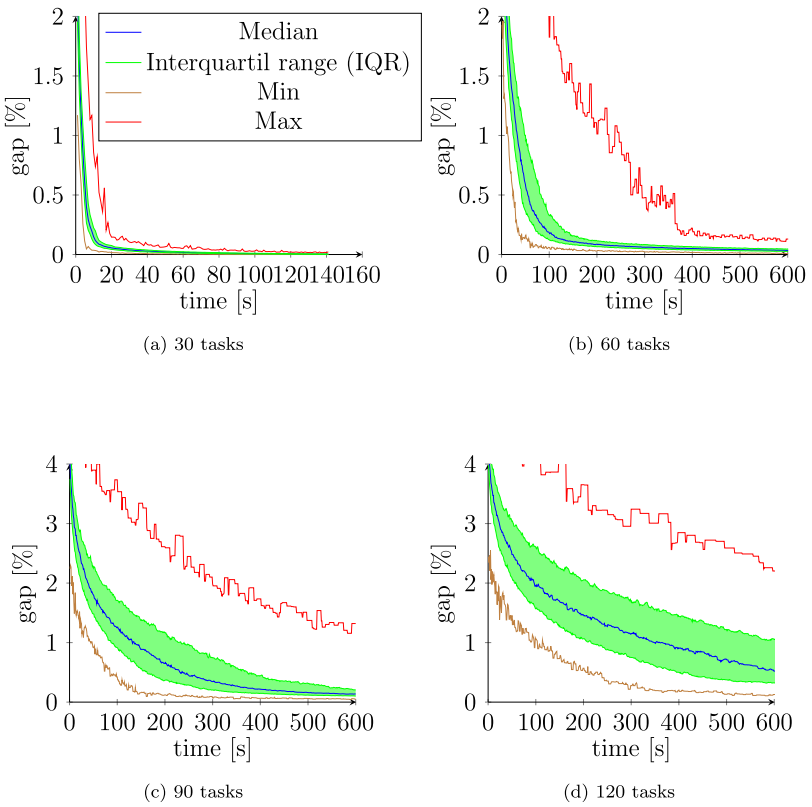
#### 5.2.2. The heuristics on small instances

We only consider the instances with 30 tasks for comparing the heuristics. The exact enumeration algorithm solves these instances to optimality in under three minutes per instance and we use the optimal solution to compute the deviation of the heuristics from the optimal value. For the optimal value OPT and the result of a given heuristic H, the relative deviation, also called gap, is computed by

$$\frac{\mathrm{H} - \mathrm{OPT}}{\mathrm{OPT}}.$$

The condensed results for all values of $\Gamma$ are presented in Table 2. For each instance and heuristic, we consider the relative deviation from the optimal solution computed by the exact algorithm. In Table 2, we give the mean of these values, their standard deviation and the minimal and maximal value as well as the average runtime.

The iterative recovery heuristic performs best and is also very consistent with low standard deviations. However, it is also the slowest heuristic. The next best heuristics are the two adaptive heuristics,




(a) 30 tasks (b) 60 tasks

(c) 90 tasks (d) 120 tasks

**Fig. 3.** Convergence of the exact enumerate algorithm for 10 min.

the adaptive McCormick and the adaptive single-stage heuristic. Unsurprisingly, they perform better than their static counterparts with fixed Values of $M$ and $\Gamma$ respectively. Between the two, the adaptive McCormick heuristic performs slightly better while taking nearly four times as long on average. In addition, these two heuristics are also much slower compared to the static heuristics. The four static heuristics all finish in well under a second, while the adaptive single-stage heuristic takes about 0.2 s on average and the adaptive McCormick heuristic about one second. Comparing the four static heuristics, the McCormick 0.5 heuristic with $M = 0.5 \cdot \tilde{M}$ and the single stage heuristic perform best, with McCormick being better on average and the single stage heuristic having slightly more consistent results throughout the whole test set, which is visible by the lower standard deviation.

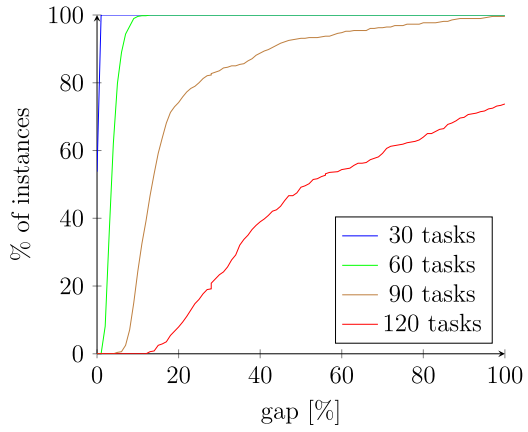



**Fig. 4.** Cumulative percentage of instances solved to within a given gap of optimality by the enumeration algorithm within 10 min sorted by size of the instances.

The McCormick heuristic with $M = 0.25 \cdot \tilde{M}$ performs worst with an average deviation of nearly 20% and overall inconsistent results visible by the comparable large standard deviation and worst case value. This may be due to $M$ being too low and restricting the choice of $x$ for most instances. This is noticeable in the fact that $x_{\max} = M$ for almost all instances. This would also explain the high standard deviation, as the solution would be very sensitive to the maximal value $x_{\max} = \max^*_{a \in A} x^*_a$ in an optimal solution $x^*$ for the computed schedule. Considering the average gap, the next best heuristic is the McCormick 1 heuristic. Comparing the results for the three versions of the McCormick heuristic seems to indicate that $M = \tilde{M}$ is chosen too high. For $M$ too large, the costs of the subproblem are overestimated. As a consequence, the schedule is chosen such that these costs are avoided and thus the schedule is very conservative. For 1412 of 1440 instances, the solution acquired by the McCormick 1 heuristic is also feasible in the worst-case scenario $t_v(\xi) = t_v + u_v$ with delay in each task. Curiously, the 38 remaining instances all share the parameter $\Gamma = 3$. This extreme conservatism in the solutions is good for high values of $\Gamma$, since it guard against uncertainties. However, it is unnecessary for small values of $\Gamma$ and results in bad solutions. This also causes the high standard deviation in this heuristic. The differences in the three versions of the McCormick heuristic with fixed $M$ also highlight how important a good choice of $M$ is and why the adaptive McCormick heuristic performs much better.

To further discuss this behavior, we will now compare the results for different values of $\Gamma$. Fig. 5 plots the percentage of instances solved within a given deviation from the optimal value for the different values of $\Gamma$ in a Performance Profile. Note, that for higher values of $\Gamma$, and thus a higher number of delayed tasks, the optimal value of the exact solution is higher. Also, for a given baseline schedule, the optimal value of the second stage problem, namely the costs paid for deviating from the baseline schedule, will be higher too.

We will now again discuss the McCormick 1 heuristic. For $\Gamma = 5$ and $\Gamma = 10$, the heuristic computes the optimal solution for the

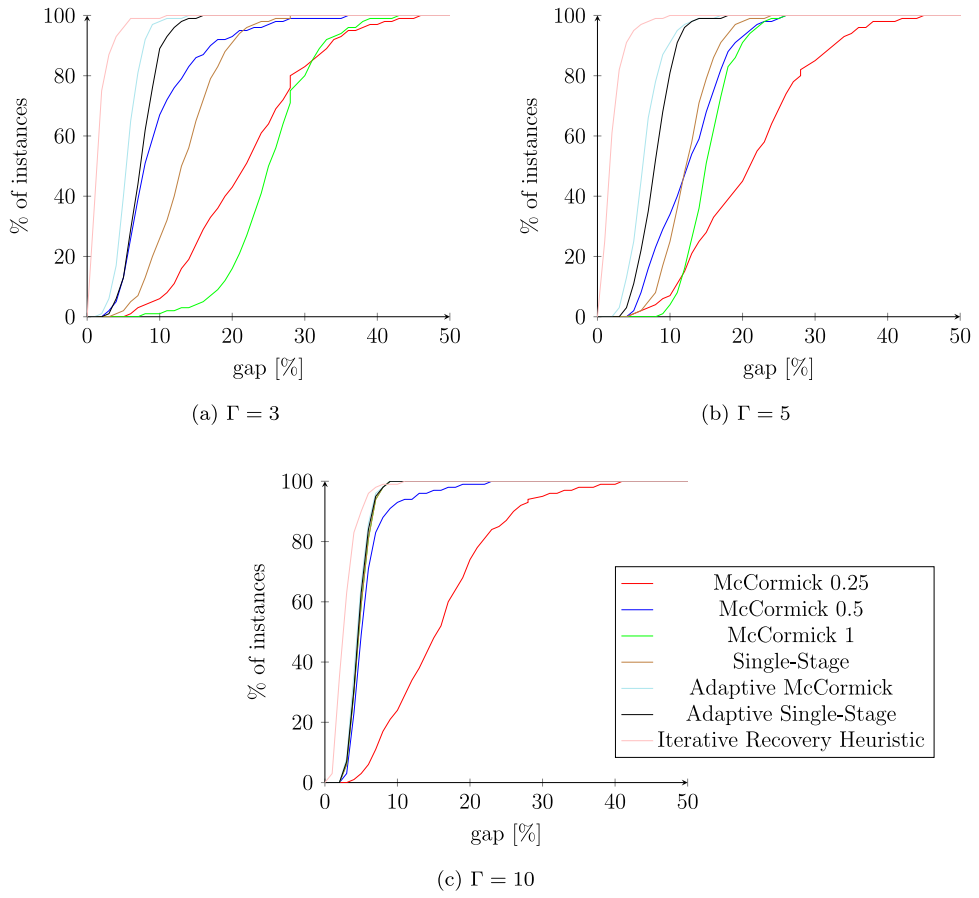



**Fig. 5.** Cumulative percentage of instances solved to within a given gap of optimality for instances with 30 tasks and varying values $\Gamma$.

worst-case scenario $t_v(\xi) = t_v + u_v$ with delay in each task. This is a consequence of the subproblem overestimating the costs and thus the schedule being chosen extremely conservative. It is important to note that for $\Gamma \in \{5, 10\}$, the heuristic computes the same solutions independent of $\Gamma$ and the difference in the plot are due to change in the optimal solution, which is more conservative for higher values of $\Gamma$. Therefore, the solution computed by the McCormick 1 heuristic is better in comparison to the optimal solution even though it is unchanged for most instances. Because of this, the optimal solution of the subproblem, and thus the cost for deviating from the schedule, are zero for these instances. For this reason, the single-stage heuristic is better than McCormick 1 for every $\Gamma$. It also computes a guaranteed solution but does not overestimate the value of $\Gamma$. The plot also shows that these two heuristics compute very similar results for $\Gamma = 10$. Because the instances all have 30 tasks, a large portion of tasks may be delayed. Also, for most tasks each possible critical path as defined in Section 3 has length less than 10. Thus, each task on the path can be delayed for $\Gamma = 10$. Thus, the solution of the single-stage problem has to be feasible for the worst-case scenario in which all tasks are delayed. Here, this is the case for 477 of 480 instances.

To conclude this comparison, the McCormick 1 heuristic overestimates $\Gamma$ and the costs for deviating from the schedule and thus computes solutions that are feasible not only for every scenario considered by $\Gamma$ robustness but also for the worst-case scenario. Therefore, it is recommended to use a different value of $M$ for lower values of $\Gamma$.

Next, we consider the remaining two variants of the McCormick heuristic with fixed parameter $M$. Again, the McCormick 0.25 heuristic performs worst but still improves with rising $\Gamma$. This is due to the more conservative solutions for higher values of $\Gamma$ causing lower $x_{\max}$ in the optimal solutions which are closer to the upper bound $M = 0.25 \cdot \tilde{M}$ in the heuristic.

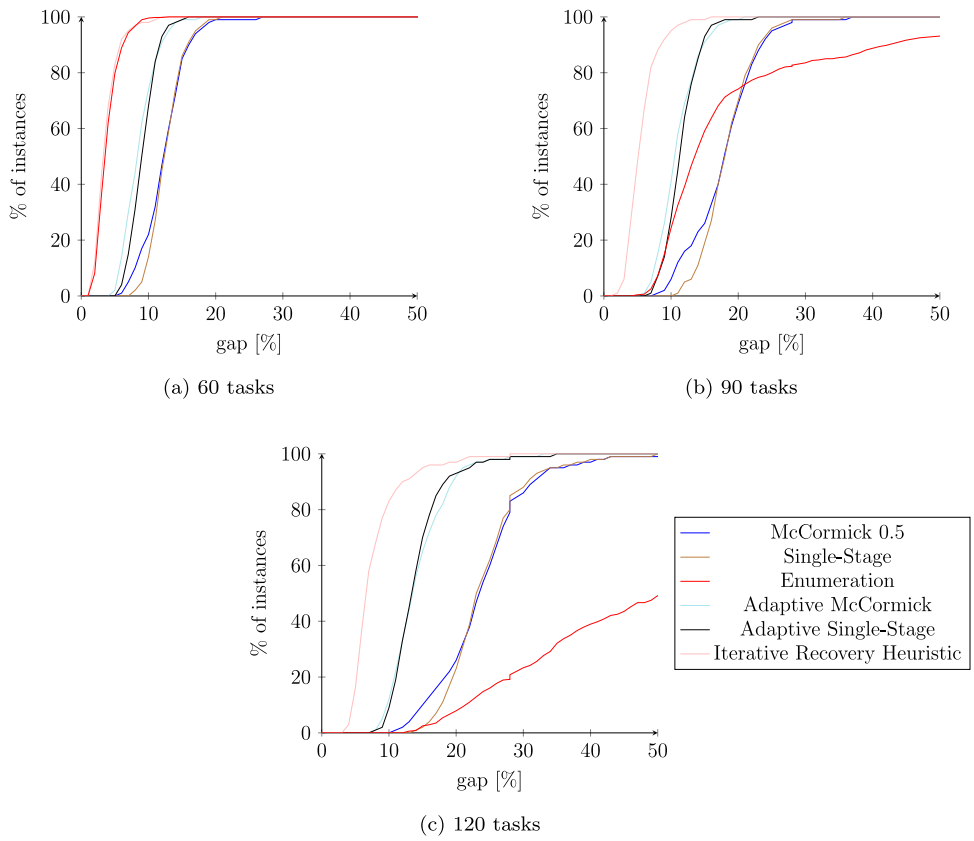
The McCormick 0.5 heuristic computes overall good results. Surprisingly, it does not improve for rising $\Gamma$ as the other heuristics do. It performs worst for $\Gamma = 5$ and best for $\Gamma = 10$. However, it is still the second worst heuristic for $\Gamma = 10$. In that case, more conservative solutions seem to be better and this choice of $M$ underestimates the costs for deviating from the baseline schedule.

Next, we consider both adaptive heuristics. For each value of $\Gamma$, they outperform the other, static heuristics. For the McCormick heuristic, this demonstrates the importance of choosing the right value of $M$ for each instance. Similar to the McCormick 0.5 heuristic, both adaptive heuristics perform best for $\Gamma = 10$ and worst for $\Gamma = 5$. For $\Gamma = 3$, the adaptive McCormick heuristic is much better than the Adaptive Single-Stage heuristic. However, for rising $\Gamma$, this difference becomes smaller until for $\Gamma = 10$, both heuristics perform nearly the same.

Finally, we consider the iterative recovery heuristic. Unlike the other heuristics, the solution gets worse with rising value of $\Gamma$. However, even for $\Gamma = 10$, it returns the best solution among the heuristics. It is important to remember that this heuristic is by far the slowest to compute. Thus, for time-sensitive use cases, one of the faster heuristics may be preferable even considering their inferior performance.

Comparing only the static heuristics, the McCormick 0.5 heuristic is the best choice for $\Gamma = 3$ and (barely) $\Gamma = 5$ while it is outperformed by both the single-stage and the McCormick 1 heuristic for $\Gamma = 10$.

Overall, if a very fast computation of under a second is important, the McCormick 0.5 heuristic performs best for low values of $\Gamma$ and the single-stage algorithm is preferable for high values of $\Gamma$. Since all heuristics finished in under one second per instance, it is also possible




(a) 60 tasks (b) 90 tasks

(c) 120 tasks

**Fig. 6.** Cumulative percentage of instances solved to within a given gap of optimality for different sizes of instances.

**Table 3**
Runtime of the heuristics on larger instances.

| Heuristics | 60 tasks [s] | 90 tasks [s] | 120 tasks [s] |
|---|---|---|---|
| McCormick 0.5 | 0.090 | 0.144 | 1.096 |
| Single-stage | 0.010 | 0.025 | 0.049 |
| Adaptive McCormick | 2.510 | 4.247 | 6.482 |
| Adaptive single-stage | 1.119 | 2.164 | 3.559 |
| Iterative recovery heuristic | 4.033 | 7.206 | 11.505 |
| Enumeration algorithm | 600 | 600 | 600 |

to compute both results and pick the better schedule. If the computations can take a few seconds, depending on the time available, using the adaptive McCormick heuristic or the Iterative Recovery heuristic is recommended for this size of instances.

### 5.2.3. Comparison on larger instances

Finally, we want to compare the results of the enumeration algorithm with a time limit of 10 min to the results of the heuristics for instances with 60, 90 and 120 tasks. In this section, we do not consider the McCormick 1 and the McCormick 0.25 heuristics due to their comparatively bad performance. We used $\Gamma = 10$ for all instances. The runtimes of the heuristics are given in Table 3.

As seen for smaller instances, the two fixed heuristics are much faster than the adaptive heuristics and the iterative recovery heuristic. However, they all compute a solution much faster than the 10 min given to the enumeration algorithm. The results are shown in Fig. 6. Similar to Fig. 4, it shows the cumulative percentage of instances solved to within a given gap of optimality. However, for these instances, we do not know an optimal solution. Thus, all gaps are computes using the dual bound from the enumeration algorithm. Since this is only a lower bound on the optimal solution, the gaps to the optimal solutions might be smaller than shown here.

The smallest instances with 30 tasks were solved to optimality by the enumeration algorithm in under 10 min. However, for larger instances, the solution found within the time limit becomes worse compared to the solutions found by the heuristics, especially compared to the iterative recovery heuristic. The two algorithms return similar results on instances with 60 tasks, but for larger instances, the iterative recovery heuristic returns much better results in the same time. For the largest instances, even the two fixed heuristics return better results than the enumeration algorithm. On these instances, we can also clearly see the differences between the iterative recovery heuristic, the adaptive heuristics and the fixed heuristics. The differences between these three classes of heuristics only become larger with increasing size of the instances.

For instances with 60 tasks, the iterative recovery heuristic solves all instances with a maximal gap of 12% which is only slightly better than the 13% gap reached by the enumeration algorithm on all instances. Compared to this, the adaptive heuristics reach this gap in only about 95% of the instances, while the fixed heuristics solve only 60% of the instances within this 13% gap. However, the enumeration algorithm took 10 min to compute the solution of one instance, while the heuristics were much faster.

Considering the instances with 90 tasks, the enumeration algorithm does no longer yield better results than the adaptive heuristics and sometimes, it performs even worse than the fixed heuristics. The corresponding red graph in the top right figure starts off steeper than those for the fixed heuristics. The algorithm still solves 74% of the instances within a gap of 20% while the McCormick 0.5 and the Single Stage heuristic solve 61% and 63% of the instances within this gap, respectively. However, the remaining instances were solved with a

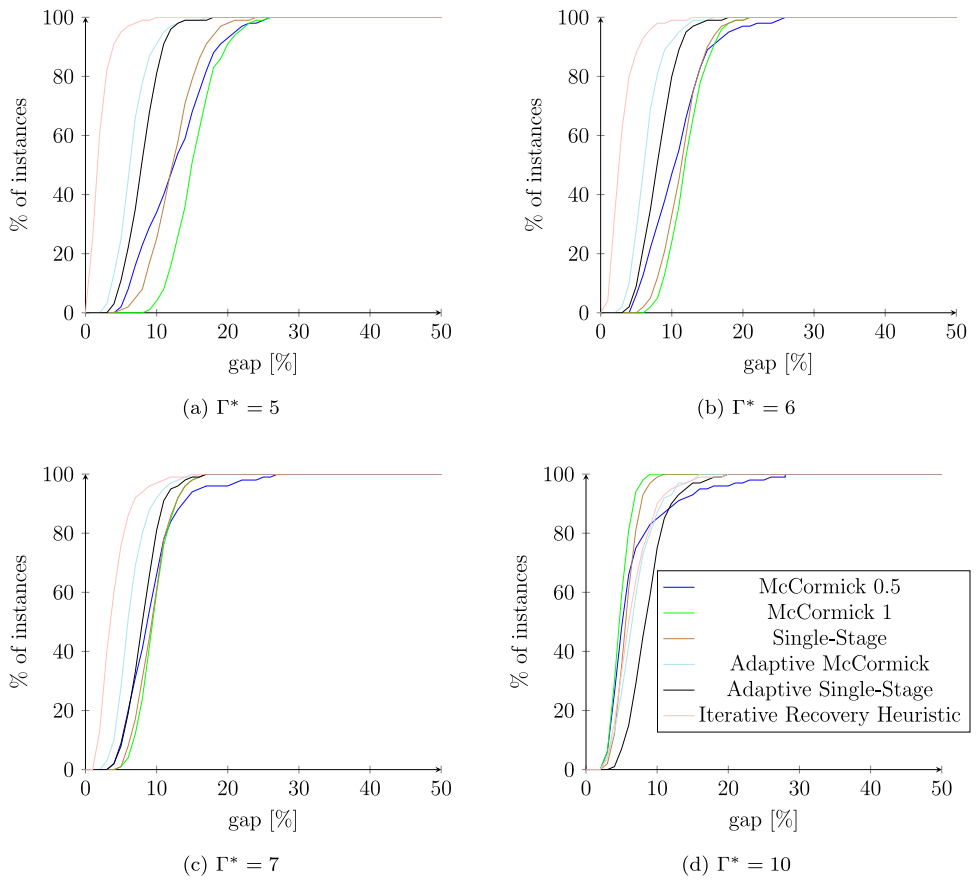



**Fig. 7.** Cumulative percentage of instances solved to within a given gap of optimality for instances with 30 tasks and varying values $\Gamma^*$ using the baseline schedule computed for $\Gamma = 5$.

larger gap by the enumeration algorithm. Stated otherwise, the fixed heuristics solve every instance within a gap of 31%. The enumeration algorithm solves less than 85% of the instances within this gap. It can also be seen that the red graph rises far less steep from this point onward.

For the largest instances with 120 tasks, the enumeration algorithm with a time limit of 10 min performs worse than each of the heuristics. This further indicates that the time limit was set too short for the large instances.

Overall, we can summarize that the enumeration algorithm with time constraints outperforms the heuristics when time is available to solve the problem to optimality. However, for larger instances, a runtime of 10 min is not sufficient to obtain solutions of similar quality as the heuristics. The same recommendations as for smaller instances apply to the choice of heuristic depending on the time available.

### 5.3. Robustness

In this section, we test robustness of the computed baseline schedules against wrong choices of $\Gamma$. To this end, for each of the small instances with 30 tasks and heuristic, we compute the baseline schedule using $\Gamma = 5$. Then, we use this schedule and compute the adjustment costs from the second-stage problem for $\Gamma^* \in \{6, 7, 10\}$. The results are displayed in Fig. 7.

The gap in this Figure refers to the gap between the objective value of the baseline schedule computed for $\Gamma = 5$ and the optimal baseline schedule for $\Gamma^*$. We added the graph for $\Gamma = \Gamma^* = 5$ to serve as a comparison.

We can see that the heuristics react differently to the changes in $\Gamma^*$. While the solution computed by the iterative recovery heuristic becomes worse with rising $\Gamma^*$, that of the adaptive McCormick heuristics barely changes, while the schedules computed by the McCormick 1 and the single-stage heuristic become better with rising $\Gamma^*$.

Interestingly, the baseline schedules computed by the McCormick 0.5 heuristic seem to become better with rising $\Gamma^*$ for some instances, while they becomes worse for other instances. This can be seen in the curve being steeper in the beginning and more flattened later for $\Gamma^* = 10$. This would again indicate that, for good choices of $\Gamma$, $M = 0.5 \cdot \tilde{M}$ is the best choice for a fixed McCormick heuristic.

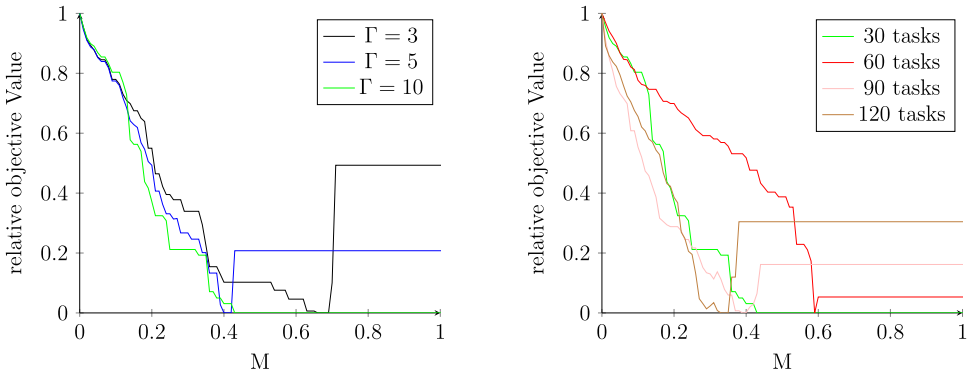
The better results for higher values of $\Gamma^*$ in the McCormick 1 and single-stage heuristic are due to the very conservative baseline schedule we already mentioned in Section 5.2.2. With rising $\Gamma^*$, the optimal solution will become more conservative and thus closer to the baseline schedules computed by the heuristic. This is why the McCormick 1 heuristic computes the best results for $\Gamma^* = 10$.

It is surprising that the quality of the baseline schedule computed by the adaptive McCormick heuristic changes so little with increasing $\Gamma$. This indicates that the value of $\Gamma$ has little effect on the choice of the best value for $M$.

The adaptive single-stage heuristic performs relatively similar for $\Gamma \in 5, 6, 7$. However, it is much worse on $\Gamma = 10$. It seems like for $\Gamma = 10$ a much more conservative solution is required than for $\Gamma = 7$.

The iterative recovery heuristic performs as expected, with the computed baseline schedule becoming worse for higher values of $\Gamma$. However, it is important to note that this heuristic still performs best in the cases $\Gamma = 6$ and $\Gamma = 7$. The schedules computed by both the McCormick 1 heuristic and the single-stage heuristic outperform the iterative recovery heuristic only for large changes in $\Gamma$.

To summarize, the algorithms that compute more conservative solutions perform better when the values of $\Gamma$ are chosen poorly. However,




(a) 30 tasks, different Gammas

(b) differently sized instances, $\Gamma = 10$

**Fig. A.8.** Relative objective value of the solutions computed by the McCormick heuristic for different values of $M$.

this alone does not justify the loss of quality when $\Gamma$ is chosen close to the realized scenario. Overall, all heuristics seem to be robust to small changes in $\Gamma$.

## 6. Conclusion

In this paper, we have introduced a new formulations for the $\Gamma$-robust two-stage project scheduling problem, a linear program with exponential many constraints, which resulted in an exact exponential-time enumeration algorithm. We also proved the NP-hardness of the second-stage problem. We then derived several heuristics, the McCormick heuristic from this formulation, the single-stage heuristic from an optimal algorithm for the $\Gamma$-robust single-stage project scheduling problem, and the iterative recovery heuristic. The enumeration algorithm and the heuristics were then tested on benchmark instances, which showed that the enumeration algorithm with a time limit set to at most 10 min is a good choice for small instances with up to 60 tasks, if time is available. For larger instances, however, the heuristics return better results. Finally, we were able to demonstrate the robustness of the heuristics against wrong choices of $\Gamma$. For future research, transferring these results to the RCPSP and conducting a computational study may show interesting results.

## CRediT authorship contribution statement

**Arie M.C.A. Koster:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Supervision, Validation, Methodology, Writing – review & editing. **Jenny Segschneider:** Conceptualization, Data curation, Formal analysis, Investigation, Validation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Nicole Ventsch:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Visualization, Writing – original draft.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Appendix. Choosing the best M for the McCormick heuristic

Remember, that

$$x_a \leq \sum_{v \in V} d_v^+ =: \tilde{M}.$$

and we consider $M_a := a \cdot \tilde{M}$ for $a \in [0, 1]$ and refer to the McCormick heuristic using this parameter as McCormick $a$. In Fig. A.8, we can see the relative objective value of the solution computed by the McCormick $a$ heuristics for $a \in [0, 1]$ for different numbers of tasks and values of $\Gamma$ for one instance per number of tasks.

There, we can see that each graph has a similar form of first falling to a minimum value at around $a \in (0.4, 0.8)$ and then rising until it eventually plateaus. However, this seems to be the only consistency in this experiment. We use this form to define the parameters of the adaptive McCormick heuristic. With this, we can also be reasonably sure to always find a global optimum with a simple descent method.

At first glance, it seems like the plateau is higher for lower values of $\Gamma$ compared to the number of tasks. In the instance with 30 tasks and $\Gamma = 10$, a third of all tasks is delayed and the plateau is close to the optimal solution. Conversely, in the instance with 30 tasks and $\Gamma = 3$, the plateau is much higher. However, this does not hold when comparing the plateaus for the instance with 30 tasks and $\Gamma = 3$ with the instance with 120 tasks and $\Gamma = 10$. Similarly, the minimum of the objective value seems to be further left for smaller values of $\Gamma$ relative to the number of tasks. However, this does not hold when comparing the instance with 60 tasks to that with 30 or 120 tasks.

## References

Artigues, Christian, Leus, Roel, Nobibon, Fabrice Talla, 2013. Robust optimization for resource-constrained project scheduling with uncertain activity durations. Flex. Serv. Manuf. J. 25 (1–2), 175–205.

Artigues, Christian, Roubellat, François, 2000. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. European J. Oper. Res. 127 (2), 297–316.

Balouka, Noemie, Cohen, Izack, 2019. A robust optimization approach for the multi-mode resource-constrained project scheduling problem. European J. Oper. Res. 291.

Bartusch, Martin, Möhring, Rolf H., Radermacher, Franz J., 1988. Scheduling project networks with resource constraints and time windows. Ann. Oper. Res. 16 (1), 199–240.

Ben-Tal, Aharon, El Ghaoui, Laurent, Nemirovski, Arkadi, 2009. Robust Optimization, Vol. 28. Princeton University Press.

Bendotti, Pascale, Chrétienne, Philippe, Fouilhoux, Pierre, Quilliot, Alain, 2017. Anchored reactive and proactive solutions to the CPM-scheduling problem. European J. Oper. Res. 261 (1), 67–74.

2020. Berlin airport opens 10 years late and three times over budget. https://www.euronews.com/2020/10/31/berlin-airport-opens-10-years-late-and-three-times-over-budget.

Bertsimas, Dimitris, Sim, Melvyn, 2004. The price of robustness. Oper. Res. 52, 35–53.
Bold, Matthew, Goerigk, Marc, 2021. A compact reformulation of the two-stage robust resource-constrained project scheduling problem. Comput. Oper. Res. 130, 105232.
Bold, Matthew, Goerigk, Marc, 2022. A faster exact method for solving the robust multi-mode resource-constrained project scheduling problem. Oper. Res. Lett. 50 (5), 581–587.
Bruni, Maria Elena, Beraldi, Patrizia, Guerriero, Francesca, 2015. The stochastic resource-constrained project scheduling problem. In: Handbook on Project Management and Scheduling, Vol. 2. Springer, pp. 811–835.
Bruni, Maria Elena, Pugliese, L Di Puglia, Beraldi, Patrizia, Guerriero, Francesca, 2017. An adjustable robust optimization model for the resource-constrained project scheduling problem with uncertain activity durations. Omega 71, 66–84.
Bruni, Maria Elena, Pugliese, L Di Puglia, Beraldi, Patrizia, Guerriero, Francesca, 2018. A computational study of exact approaches for the adjustable robust resource-constrained project scheduling problem. Comput. Oper. Res. 99, 178–190.
Fahmy, Amer M., 2016. Optimization algorithms in project scheduling. Optim. Algorithms Methods Appl..
Garey, Michael R., Johnson, David S., 1979. Computers and Intractability, Vol. 174. Freeman San Francisco.
Hazır, Öncü, Ulusoy, Gündüz, 2020. A classification and review of approaches and methods for modeling uncertainty in projects. Int. J. Prod. Econ. 223, 107522.
Herroelen, Willy, Leus, Roel, 2004a. The construction of stable project baseline schedules. European J. Oper. Res. 156 (3), 550–565.
Herroelen, Willy, Leus, Roel, 2004b. Robust and reactive project scheduling: a review and classification of procedures. Int. J. Prod. Res. 42 (8), 1599–1620.
Herroelen, Willy, Leus, Roel, 2005. Project scheduling under uncertainty: Survey and research potentials. European J. Oper. Res. 165 (2), 289–306.
Holzhauser, Michael, Krumke, Sven O., Thielen, Clemens, 2016. Budget-constrained minimum cost flows. J. Combin. Optim. 31 (4), 1720–1745.
Ke, Hua, Liu, Baoding, 2010. Fuzzy project scheduling problem and its hybrid intelligent algorithm. Appl. Math. Model. 34 (2), 301–308.
Kelley, James E., Walker, Morgan R., 1959. Critical-path planning and scheduling. In: Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference. In: IRE-AIEE-ACM '59 (Eastern), Association for Computing Machinery, New York, NY, USA, pp. 160–173.
Kolisch, Rainer, Sprecher, Arno, 1997. PSPLIB - a project scheduling problem library: OR software - ORSEP operations research software exchange program. European J. Oper. Res. 96 (1), 205–216.
Liebchen, Christian, Lübbecke, Marco, Möhring, Rolf, Stiller, Sebastian, 2009. The concept of recoverable robustness, linear programming recovery, and railway applications. Robust Online Large-Scale Optim.: Models Tech. Transp. Syst. 1–27.
Minoux, Michel, 2011. On 2-stage robust LP with RHS uncertainty: complexity results and applications. J. Global Optim. 49 (3), 521–537.
Sakkout, Hani El, Wallace, Mark, 2000. Probe backtrack search for minimal perturbation in dynamic scheduling.
Terry, Tara, Epelman, Marina, Thiele, Aurélie, 2009. Robust Linear Optimization with Recourse. Technical Report, Lehigh University.
Tsoukalas, A., Mitsos, A., 2014. Multivariate McCormick relaxations. J. Global Optim. 59 (2), 633–662.
Ventsch, Nicole, 2020. $\Gamma$-robust Optimization of Project Scheduling Problems (Master's Thesis). RWTH Aachen University, Chair of Mathematics II.
Ye, Yinyu, Tse, Edison, 1989. An extension of Karmarkar's projective algorithm for convex quadratic programming. Math. Program. 44, 157–179.
Zaman, Forhad, Elsayed, Saber, Sarker, Ruhul, Essam, Daryl, Coello, Carlos A Coello, 2021. Pro-reactive approach for project scheduling under unpredictable disruptions. IEEE Trans. Cybern. 52 (11), 11299–11312.
Zhu, Guidong, Bard, Jonathan, Yu, Gang, 2007. A two-stage stochastic programming approach for project planning with uncertain activity durations. J. Sched. 10, 167–180.