Γ-robust Optimization of Project Scheduling Problems

Arie M.C.A. Koster [*] Jenny Segschneider [†] Nicole Ventsch[‡]

December 19, 2022

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 estimation together with an interval in which this time can vary is given. At most Γ of the jobs will deviate from the estimated time.
We present two solution approaches to this problem. The first approach treats the problem of determining 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 determining the baseline schedule in the first stage and an adaptation to the scenario in the second stage. We introduce a novel formulation and derive an exact algorithm and two heuristics. A computational study on benchmark instances reveals that the heuristics perform well on larger instances.

[*]RWTH Aachen University, Research Area Discrete Optimization
E-Mail: koster@math2.rwth-aachen.de

[†]RWTH Aachen University, Research Area Discrete Optimization
E-Mail: segschneider@math2.rwth-aachen.de

[‡]RWTH Aachen University

1 Introduction

The 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 [9] for an overview of different models and algorithms solving PSP. In this paper, we consider two variations of PSP without resource constraints. If all parameters are known with certainty, this problem can be solved easily using the critical path method introduced by Kelley & Walker [14]. 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, many research efforts aimed at modeling the reality more precisely by taking uncertainty into consideration. Herroelen and Leus [12] 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. For a survey on robust optimization, see for example Ben-Tal et al [3].

In robust optimization, scenarios from a predefined uncertainty set are considered. In this paper, we will consider the so-called budget uncertainty, also known as Γ-robustness, proposed by Bertsimas & Sim [4] for two variants of the PSP, the single stage and the two-stage problem.

The Γ-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. Bruni et al. [7] found a similar problem with a different objective in the subproblem of their two-stage formulation. They introduce 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 & Goerigk [5] who proposed a compact LP formulation based on the algorithm. The Γ-robust two-stage problem is based on the formulation from Zhu et al. [20]. They propose a model which balances the cost of the baseline schedule and the expected cost for deviations from this schedule using stochastic scheduling. Instead, we employ robust scheduling to determine the schedule with lowest costs in the worst-case scenario.

Robust optimization approaches as defined by Ben-Tal et al. [3] have been rarely used for PSP, primarily in resource constrained project scheduling. For example, Artigues et al. [1] developed a scenario-relaxation algorithm and were the first to apply robust optimization to PSP. Furthermore, Bruni et al [6] proposed a chance-constraint based heuristic and a two-stage adjustable robust optimization model [7]. They [8] also compare this algorithm against an additional Benders-style algorithm in a computational study. Most recently, Bold & Goerigk [5] introduced a compact reformulation of the two-stage robust resource-constrained PSP.

However, there are different approaches often referred to as robust. Herroelen & Leus [10] 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 & Liu [13] 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 & Leus [11].

The contributions of this paper are twofold. First, we introduce a new derivation for the Γ-robust Single-Stage PSP which can be solved using the algorithm from Bruni et al. [7]. Second, we propose a robust approach for the stochastic model by Zhu et al [20] and create a novel formulation. For the resulting problem, an exact algorithm and two heuristics are proposed and tested in a computational study.

This paper is organized as follows. In the next section, we introduce the single-stage project scheduling problem and compare it to the problem introduced by Bruni et al. [7]. Subsequently, we will revisit their findings and transfer them to the single-stage PSP. Section 3 treats the Γ-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 [18] and gives us an upper bound for the optimal solution. Moreover, we develop a polynomial heuristic based on the single-stage algorithm. In Section 4, 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 [?].

2 The Γ-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.

2.1 Budget uncertainty

For the Γ-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 Γ tasks will not finish in the estimated time following the ideas by Bertsimas and Sim [4]. Therefore, the so-called uncertainty set is given by

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

and the time needed for all activities $v \in V$ 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 Γ 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 Γ 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 [16].

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:

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

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:

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

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

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

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

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

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

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

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

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 Γ-robustness as defined above.

The violated paths resemble the critical paths defined by, among others, Kelley and Walker [14]. 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.

2.2 Relation to RCPSP

A similar problem was first examined by Bruni et al [7, Section 5]. They introduce a new formulation for the two stage resource constraint PSP (TSRCPSP) where the subproblem, after being rewritten and linearized in Section 5, resembles 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. [7]. 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. [2] 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

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

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 ξ.

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. [7] can also be applied to solve our problem.

2.3 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 $p_v^0, \ldots, p_v^\Gamma$ from the source node to v, where each path p_v^γ includes exactly $\gamma \le \Gamma$ delays. For a node v_j, the longest path with exactly γ 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 γ delays as $F_{v_j,\gamma}$. These considerations lead to the following recursive algorithm.

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

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 $\Gamma \leq |V|$, we know that the complexity is at most $\mathcal{O}(|V|^3)$, so that this algorithm runs in polynomial time.

3 The Γ-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 [20].

Additionally to the notation used in Section 2, we need the following definitions:

- $c_v \geq 0$ is the cost for task $v \in V$ per time unit; we set $c_{v_0} := c_{v_{n+1}} := 0$
- $y_v^+ \geq 0$ 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)
- $y_v^- \geq 0$ 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)
- $d_v^+ \geq 0$ is the cost per time unit for finishing task $v \in V$ later than in the formerly set project schedule (F_v)
- $d_v^- \geq 0$ 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 $c^T F = \sum_{v \in V} c_v \cdot F_v$, 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 ω are known, the deviations $y_v^+, y_v^- \geq 0$ have to satisfy the following condition:

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

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 ω is defined by vector ξ):

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

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:

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

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

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

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

3.1 The Second Stage problem

In order to solve the problem, we will now introduce a compact formulation for the second stage problem.

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

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

◇

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

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

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 ξ and x. By substituting ξ for $\xi^+ - \xi^-$ to deal with the inequality constraint $\sum_{v \in V} |\xi_v| \leq \Gamma$, we can formulate this problem as

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

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 & Tse [19]. 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. [17].

Let X be the set of feasible dual variables

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

and

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

the new uncertainty set. We assume that there is an enumeration of the extreme points of X, i.e., a set $\tilde{X} = \{x^i \colon 1 \leq i \leq K\} \subseteq \mathbb{R}^{|A|}_{\geq 0}$ for some $K \in \mathbb{N}$. For ξ 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).

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

Note, that for each $k\in K$ there is an inner maximization problem of finding the worst-case scenario in Φ. 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:

$$\begin{aligned} &\min_{\theta^k,\lambda^{k,+},\lambda^{k,-}} \Gamma\theta^k + \sum_{v\in V}(\lambda_v^{k,+} + \lambda_v^{k,-}) \\ &\text{s.t. } \theta^k + \lambda_v^{k,+} \geq \sum_{w\in V:\,(w,v)\in A} u_v \cdot x_{(w,v)}^k && \forall v\in V \\ &\qquad \theta^k + \lambda_v^{k,-} \geq -\sum_{w\in V:\,(w,v)\in A} u_v \cdot x_{(w,v)}^k && \forall v\in V \\ &\qquad \theta^k, \lambda^{k,+}, \lambda^{k,-} \geq 0 \end{aligned}$$

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

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

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.

3.2 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^*, Ω^*, 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 Ω. This is equivalent to computing

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

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 Tsoukalar & Mitsos [18]. 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 $\tilde{\mathcal{U}}_\Gamma$ defines a polyhedron. For fixed x, the maximization problem is linear in $\xi^+, \xi^- \in \tilde{\mathcal{U}}_\Gamma$ 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 linearise 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:

$$\begin{aligned} 0 \le q^+_a, q^-_a &\le x_a && \forall a \in A \\ M \cdot \xi^+_w + x_{(v,w)} - M \le q^+_{(v,w)} &\le M \cdot \xi^+_w && \forall (v,w) \in A \\ M \cdot \xi^-_w + x_{(v,w)} - M \le q^-_{(v,w)} &\le M \cdot \xi^-_w && \forall (v,w) \in A \end{aligned}$$

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

$$
\begin{aligned}
\max_{x,\xi^+,\xi^-} \quad & \sum_{a=(v,w)\in A} (F_v - F_w + t_w)x_a + q_a^+ \cdot u_w - q_a^- \cdot u_w \\
\text{s.t.} \quad & \sum_{a=(\cdot,v)\in A} x_a - \sum_{a=(v,\cdot)\in A} x_a \le d_v^+ && \forall\, v \in V\backslash\{v_0\} \\
& -\sum_{a=(\cdot,v)\in A} x_a + \sum_{a=(v,\cdot)\in A} x_a \le d_v^- && \forall\, v \in V\backslash\{v_0\} \\
& x_a \ge 0 && \forall\, a \in A \\
& \sum_{v\in V} \xi_v^+ + \xi_v^- \le \Gamma \\
& q_a^+, q_a^- \le x_a && \forall a \in A \\
& q_{(v,w)}^+ \le M \cdot \xi_w^+ && \forall (v,w) \in A \\
& q_{(v,w)}^- \le M \cdot \xi_w^- && \forall (v,w) \in A \\
& q_{(v,w)}^+ \ge M \cdot \xi_w^+ + x_{(v,w)} - M && \forall (v,w) \in A \\
& q_{(v,w)}^- \ge M \cdot \xi_w^- + x_{(v,w)} - M && \forall (v,w) \in A \\
& x_a, q_a^+, q_a^- \ge 0 && \forall a \in A \\
& \xi_v^+, \xi_v^- \in \{0,1\} && \forall\, v \in V
\end{aligned} \tag{10}
$$

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^*, Ω^*, a new vertex x^* is computed by solving (10). If the objective is greater than Ω^*, a violated constraint is found and we add x^* to $\hat{X}$. Else, the solution F^*, Ω^* is optimal.

Algorithm 1 Exact algorithm for the Γ-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 Γ-robust two-stage problem and the finish times for all tasks $(F_v)_{v \in V}$

```
 1: while true do
 2:     $schedule\_obj, F^*, \Omega^*$ optimal solution of (9) with constraints only for $\hat{X}$
 3:     $subproblem\_obj, x^*$ = optimal solution of (10)
 4:     if $subproblem\_obj > \Omega$ then
 5:         $\tilde{X} = \tilde{X} \cup \{x\}$
 6:     else
 7:         break;
 8:     end if
 9: end while
10: return $schedule\_obj, (F_v)_{v \in V}$
```

Algorithm 1 finds an optimal solution for LP (9) and therefore solves the Γ-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 ε.

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.

3.3 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

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

Now, we have a compact formulation as a linear program that we can solve efficiently and which produces a baseline schedule. However, we should keep in mind that this formulation is based on a relaxation.

- 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 Γ 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.

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. To summarize, the complete heuristic based on McCormick consists of first computing a baseline schedule F^* by solving (11) optimally and then calculation its cost by solving (10) for F^*.

In order to get the actual robust deviation costs for the produced schedule, a mixed integer program is used. Thus, the algorithm cannot necessarily be solved efficiently. 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.

3.4 Single-Stage Heuristic

In Section 2, 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 Γ-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 2, 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.

4 Computational Study

Based on the algorithms presented in the previous sections, we conduct a computational study. First, we compare the McCormick heuristic introduced in Section 3.3 for three different values of M and the single-stage heuristic from Section 3.4. Next, we analyze the convergence of the enumeration algorithm (1) with on instances with different sizes. Lastly, we compare the results of the enumeration algorithm with restricted runtime to the heuristics.

All three 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, [15]). All models were coded in Python and solved using Gurobi 9.1.2, running on an Intel Core i5-3570 CPU 3.4GHz RAM 16GB.

4.1 Instances

For each instance, three parameters are given to measure its difficulty, two of which refer to the resource constraint 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 Γ. 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 the exact algorithm and the McCormick heuristic. From the interpretation of the subproblem in Section 3.1, 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 precise the closer the bound provided by M is, we choose different values and compare their results. More precisely, we consider $M_a := a \cdot \tilde{M}$ for $a \in \{1, 0.5, 0.25\}$ and refer to the McCormick heuristic

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
$\left(\tilde{d_v^+}\right)_{v\in V\setminus\{v_0\}}$	5	10

Table 1: Bounds for the parameter settings

using this parameter as McCormick a. This leads to three versions of the McCormick heuristic and thus four heuristics and one exact algorithm in total.

4.2 Results

In this section, we first present the overall performance of the four heuristics for all instances. Then, we will evaluate the influence of different values of Γ. Next, we will consider the runtime and convergence properties of the exact algorithm. Last, we will compare the results of the heuristics to that of the enumeration algorithm with a runtime restriction of 10 minutes.

4.2.1 The Heuristics

We only consider the instances with 30 activities 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 is computed by

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

The condensed results for all values of Γ and NC 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.

Comparing the four heuristics, the McCormick heuristic with $M = 0.5 \cdot \tilde{M}$ and the single stage heuristic perform best, with McCormick being better on average and the

heuristic	Mean	SD	Min	Max
McCormick 1 [%]	15.01	3, 80	1.49	42.55
McCormick 0.5 [%]	9.21	4.32	1.53	35.95
McCormick 0.25 [%]	19.48	8.04	1.82	45.15
single-stage [%]	10.15	2.95	1.49	28.49

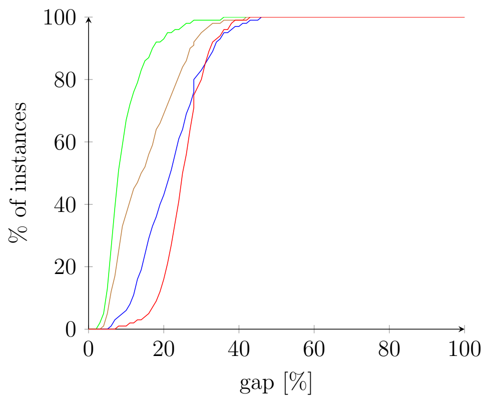
Table 2: Deviation from the optimal value for the instances with 30 tasks.

single stage heuristic having more consistent results throughout the whole test set, which is visible by the lower standard deviation.

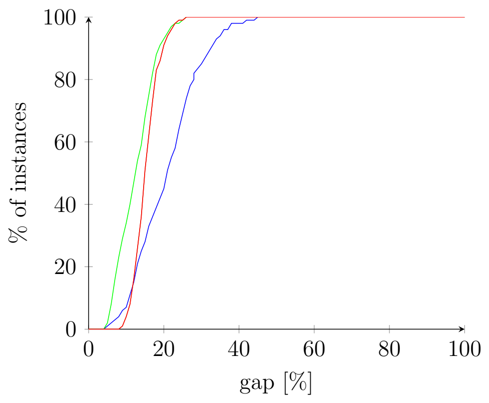
The McCormick heuristic with $M = 0.25 \cdot \tilde{M}$ performs worst with an average deviation of nearly 20% and overall inconsistent results visible by the comparable large standard deviation and worst case value. This may be due to M being too low and restricting the choice of x for most instances. 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 deviation, 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 $\xi = \begin{Bmatrix} 1 & \dots & 1 \end{Bmatrix}$ with delay in each task. Curiously, the 38 remaining instances all share the parameter $\Gamma = 3$. To further discuss this behavior, we will now compare the results for different values of Γ.

Figure 1 plots the percentage of instances solved within a given deviation from the optimal value for the different values of Gamma. First, we discuss some properties of the optimal solution and the subproblem of the McCormick heuristic depending on Γ. Obviously, for higher values of Γ, and thus a higher number of delayed tasks, the optimal value is higher. Also, for a given schedule, the optimal value of the subproblem, namely the costs paid for deviating from it, will be higher too. This may cause the McCormick heuristic to compute more robust solutions and thus lead to lower costs in the subproblem for the optimal schedule.

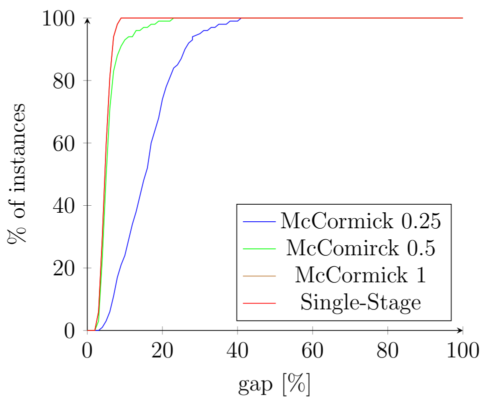
We will now again discuss the McCormick 1 heuristic. For $\Gamma = 5$ and $\Gamma = 10$, the heuristic computes the optimal solution for the worst-case scenario $t_v(\xi) = t_v + u_v$. This is a consequence of the subproblem overestimating the costs and thus the schedule being chosen extremely conservative. It is important to note that for $\Gamma \in \{5, 10\}$, the heuristic computes the same solutions and the difference in the plot are due to change in the



(a) $\Gamma = 3$



(b) $\Gamma = 5$



(c) $\Gamma = 10$

Figure 1: Cumulative percentage of instances solved to within a given gap of optimality for instances with 30 jobs and varying values Γ

optimal value, which is more conservative for higher values of Γ. Therefore, the solution computed by the McCormick 1 heuristic is better in comparison to the optimal solution even though it is independent of Γ 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 Γ. It also computes a guaranteed solution but does not overestimate the value of Γ. 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 2 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. Here, this is the case for 477 of 480 instances.

To conclude this comparison, the McCormick 1 heuristic overestimates Γ and the costs for deviating from the schedule and thus computes solutions that are feasible not only for every scenario considered by Γ robustness but also for the worst-case scenario. Therefore, it is recommended to use a different value for M or the single-stage heuristic, as it offers more control on the degree of conservatism in the solution through the choice of Γ.

Finally, we consider the remaining two variants of the McCormick heuristic. Again, the McCormick 0.25 heuristic performs worst but still improves with rising Γ. This is due to the more conservative solutions for higher values of Γ 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 Γ as the other heuristics do. It performs worst for $\Gamma = 5$ and best for $\Gamma = 10$.

In comparison to the other heuristics, it 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, the McCormick 0.5 heuristic performs best for low values of Γ and the single-stage algorithm is preferable for high values of Γ. Since all heuristics finished in under one second per instance, it is also possible to compute both results and pick the better schedule.

4.2.2 The Enumeration Algorithm

Next, we want to analyze the convergence and runtime of the exact algorithm. As mentioned above, for the instances with 30 jobs, 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 jobs shown in Figure 2. The blue graph represents the value of the optimal solution F^*, Ω^* 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 Ω 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 minutes. Starting with a gap of over 500%, the gap falls to under 3% within in these first 10 minutes. For comparison, after 20 Minutes, the gap reduces to about 1% and the computation finished after 3800 seconds or about 60 minutes. As expected, most progress is made in the first few minutes of

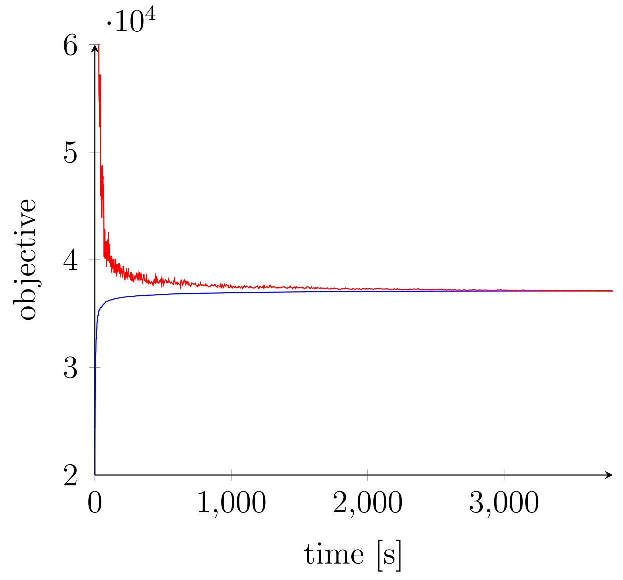


Figure 2: Convergence of the exact enumerate algorithm for one instance with 60 tasks and 10 minutes

computation. Because of this, we restricted the computation time for the enumeration algorithm to 10 minutes. In this time frame, the smallest instances with 30 jobs can be solved to optimality while the bigger instances get solved with varying gaps.

Figure 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 activities per instance. The blue graph represents the median over all instances with the corresponding number of activities and the brown and red graph the minimal or maximal gap respectively. The green space is the interquartil 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 jobs, where the gap first reduces more steeply than for the bigger instances. For instances with 30 or 60 jobs, the average gap is under 4% after 10 minutes and all instances with 30 jobs are solved to optimality within 140 seconds. For the bigger instances, the algorithm didn't compute a good result in 10 minutes and would need to run longer in order to compute a good result.

Figure 4 further illustrates the results after 10 minutes 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 jobs. The instances with 30 jobs were all solved to optimality. All instances with 60

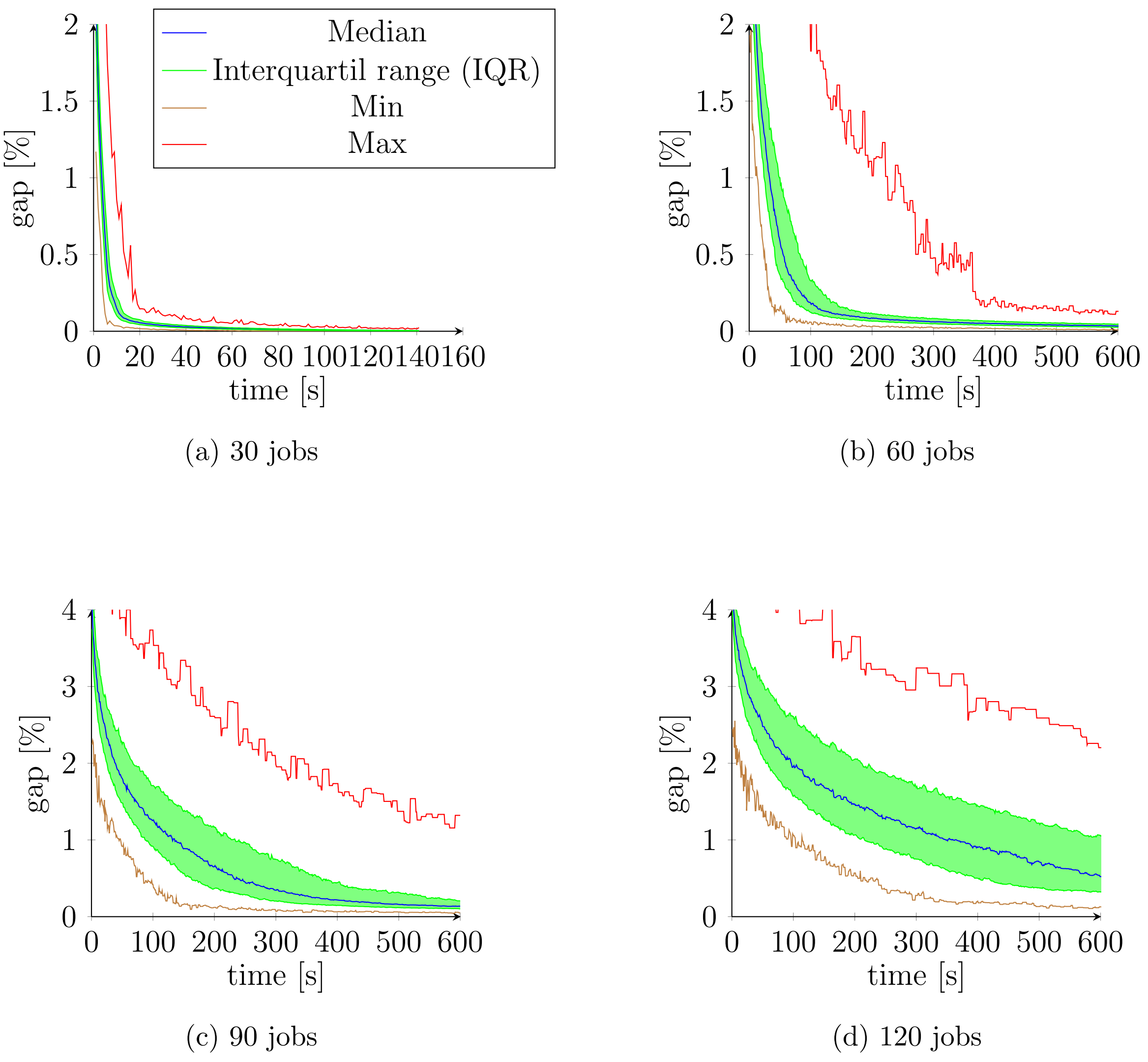


Figure 3: Convergence of the exact enumerate algorithm for 10 minutes

jobs 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 jobs, 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 jobs, only 73.75% were solved with a gap of 100% or less. We also note that the curve for the instances with 120 jobs falls less steep than the other curves. This indicates, that the time limit of 10 minutes for the algorithm was chosen too low for larger instances.

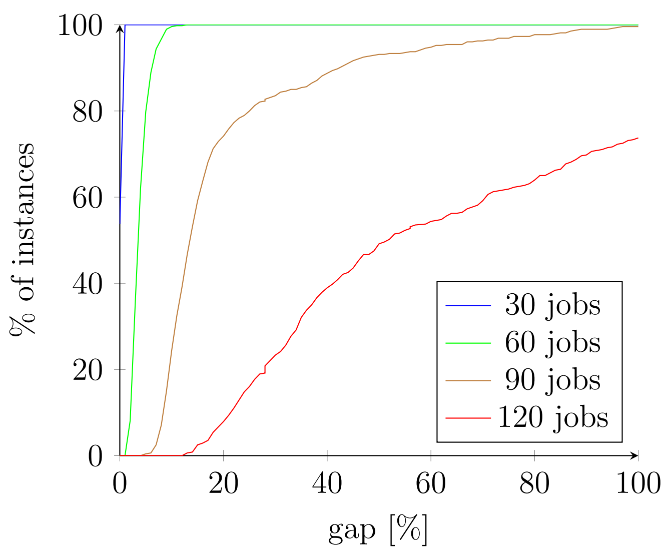


Figure 4: Cumulative percentage of instances solved to within a given gap of optimality by the enumeration algorithm within 10 minutes sorted by size of the instances

4.2.3 Comparison

Finally, we want to compare the results of the enumeration algorithm with a time limit of 10 minutes to the results of the heuristics for instances with 30, 60, 90 and 120 jobs. Because of the bad results of the McCormick 0.25 heuristic, we do not consider it in this section. We used $\Gamma = 10$ for all instances. The results are shown in Figure 5. Similar to Figure 4, it shows the cumulative percentage of instances solved to within a given gap of optimality for the McCormick 0.5, McCormick 1 and single-stage heuristics as well as the enumeration algorithm for instances with 30, 60, 90 and 120 jobs and $\Gamma = 10$.

The smallest instances with 30 jobs were solved to optimality by the enumeration algorithm in under 10 minutes, which is shown by the red graph. 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 green graph denoting the McCormick 0.5 heuristic.

For instances with 60 jobs, the enumeration algorithm solves all instances with a maximal gap of 13%. The McCormick 0.5 heuristic solves only 57% of the instances within this gap which is the best value comparing the three heuristics. However, the enumeration algorithm took 10 Minutes to compute the solution of one instance, while each heuristic needed less than a second for the larger instances.

Considering the instances with 90 jobs, the enumeration algorithm does no longer necessarily yield better results. The corresponding red graph in the bottom left figure starts of steeper than those for the heuristics. The algorithm still solves 74% of the

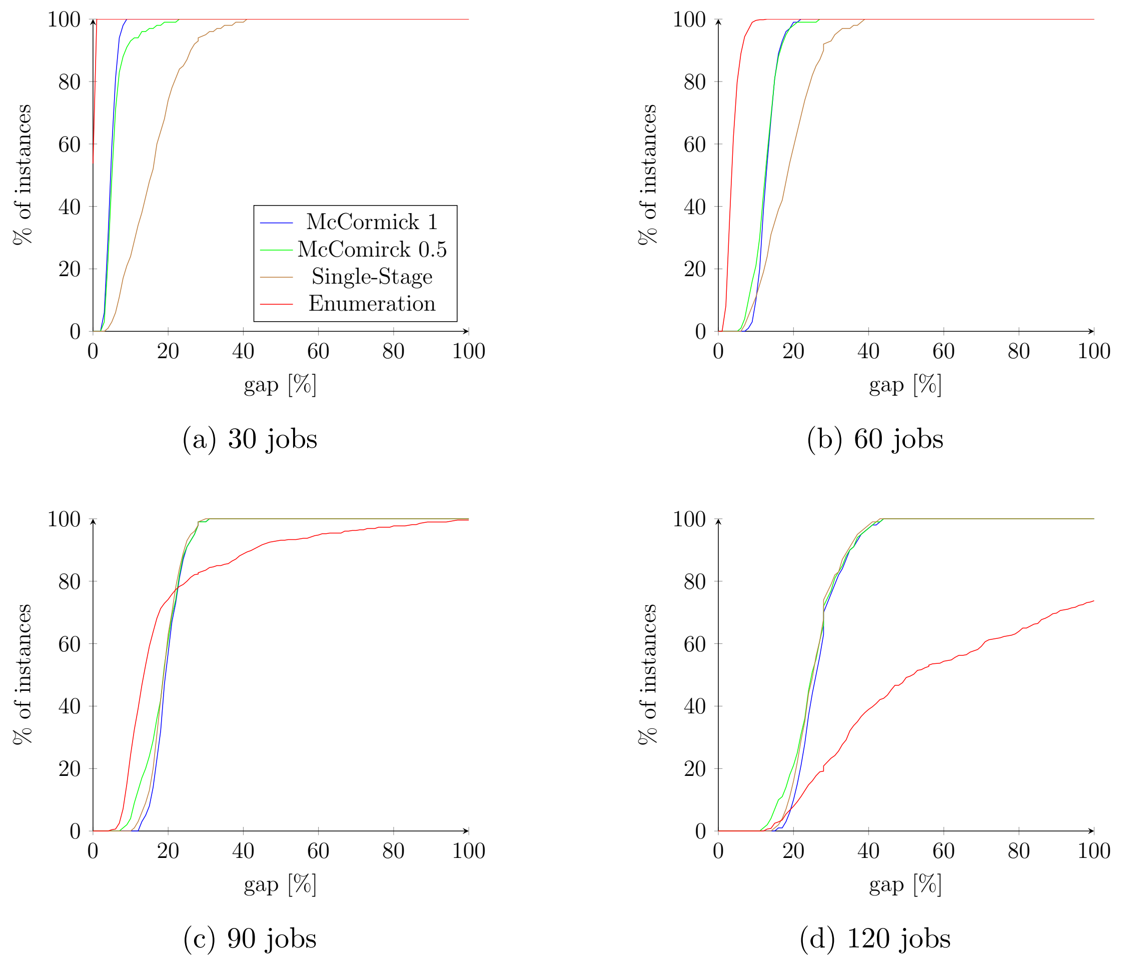


Figure 5: Cumulative percentage of instances solved to within a given gap of optimality for different sizes of instances

instances to 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 larger gap by the enumeration algorithm. The 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.

It is interesting to note that the three heuristics behave very similar for larger instances.

For the largest instances with 120 jobs, the enumeration algorithm with a time limit of 10 minutes performs worse than each of the three 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 is better than the heuristics if the time is available. However, for larger instances, a runtime of 10 minutes is not sufficient to achieve solutions with similar quality as the heuristics.

5 Conclusion

In this paper, we have introduced a new formulations for the Γ-robust two-stage project scheduling problem, a linear program with exponential many constraints, which resulted in an exact exponential-time enumeration algorithm. We also derived two heuristics, the McCormick heuristic from these formulation and the single-stage heuristic from an optimal algorithm for the Γ-robust single-stage project scheduling problem. The enumeration algorithm and the two heuristics were then tested on benchmark instances which showed that the enumeration algorithm with a time limit set to at most 10 minutes is a good choice for small instances with up to 90 jobs, if the time is available. However, for larger instances, the heuristics return better results.

Future research goals include to decide the NP-hardness of the Γ-robust two-stage project scheduling problem or finding an efficient exact algorithm for larger instances.

References

[1] Christian Artigues, Roel Leus, and Fabrice Talla Nobibon. Robust optimization for resource-constrained project scheduling with uncertain activity durations. *Flexible Services and Manufacturing Journal*, 25(1-2):175–205, 2013.

[2] Martin Bartusch, Rolf H Möhring, and Franz J Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of operations Research*, 16(1):199–240, 1988.

[3] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*, volume 28. Princeton University Press, 2009.

[4] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52:35–53, 02 2004.

[5] Matthew Bold and Marc Goerigk. A compact reformulation of the two-stage robust resource-constrained project scheduling problem. *Computers & Operations Research*, 130:105232, 2021.

[6] Maria Elena Bruni, Patrizia Beraldi, and Francesca Guerriero. The stochastic resource-constrained project scheduling problem. In *Handbook on Project Management and Scheduling Vol. 2*, pages 811–835. Springer, 2015.

[7] Maria Elena Bruni, L Di Puglia Pugliese, Patrizia Beraldi, and Francesca Guerriero. An adjustable robust optimization model for the resource-constrained project scheduling problem with uncertain activity durations. *Omega*, 71:66–84, 2017.

[8] Maria Elena Bruni, L Di Puglia Pugliese, Patrizia Beraldi, and Francesca Guerriero. A computational study of exact approaches for the adjustable robust resource-constrained project scheduling problem. *Computers & Operations Research*, 99:178–190, 2018.

[9] Amer M Fahmy. Optimization algorithms in project scheduling‖. *Optimization Algorithms-Methods and Applications*, 2016.

[10] Willy Herroelen and Roel Leus. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156(3):550 – 565, 2004.

[11] Willy Herroelen and Roel Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004.

[12] Willy Herroelen and Roel Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289 – 306, 2005.

[13] Hua Ke and Baoding Liu. Fuzzy project scheduling problem and its hybrid intelligent algorithm. *Applied Mathematical Modelling*, 34(2):301 – 308, 2010.

[14] James E. Kelley and Morgan R. Walker. Critical-path planning and scheduling. In *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM ’59 (Eastern), pages 160 – 173, New York, NY, USA, 1959. Association for Computing Machinery.

[15] Rainer Kolisch and Arno Sprecher. Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205 – 216, 1997.

[16] Michel Minoux. On 2-stage robust lp with rhs uncertainty: complexity results and applications. *Journal of Global Optimization*, 49(3):521–537, 2011.

[17] Tara Terry, Marina Epelman, and Aurélie Thiele. Robust linear optimization with recourse. *technical report, Lehigh University*, 04 2009.

[18] A. Tsoukalas and A. Mitsos. Multivariate McCormick relaxations. *Journal of Global Optimization*, 59(2):633–662, July 2014.

[19] Yinyu Ye and Edison Tse. An extension of karmarkar's projective algorithm for convex quadratic programming. *Mathematical Programming*, 44:157–179, 1989.

[20] Guidong Zhu, Jonathan Bard, and Gang Yu. A two-stage stochastic programming approach for project planning with uncertain activity durations. *J. Scheduling*, 10:167–180, 06 2007.