Robust optimization for resource-constrained project scheduling with uncertain activity durations

Christian Artigues, Roel Leus, Fabrice Talla Nobibon

Robust optimization for resource-constrained project scheduling with uncertain activity durations

Christian Artigues[ab] • Roel Leus[c] • Fabrice Talla Nobibon[c]

[a] CNRS ; LAAS
7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
artigues@laas.fr

[b] Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS
F-31077 Toulouse Cedex 4, France

[c] Research group ORSTAT, Faculty of Business and Economics
K.U.Leuven, Leuven, Belgium
Roel.Leus@econ.kuleuven.be ; Fabrice.TallaNobibon@econ.kuleuven.be

The purpose of this paper is to propose models for project scheduling when there is considerable uncertainty in the activity durations, to the extent that the decision maker cannot with confidence associate probabilities with the possible outcomes of a decision. Our modeling techniques stem from robust optimization, which is a theoretical framework that enables the decision maker to produce solutions that will have a reasonably good objective value under any likely input data scenario. We develop and implement a scenario-relaxation algorithm and a scenario-relaxation-based heuristic. The first algorithm produces optimal solutions but requires excessive running times even for medium-sized instances; the second algorithm produces high-quality solutions for medium-sized instances and outperforms two benchmark heuristics.

Keywords: project scheduling, RCPSP, uncertainty, robust optimization.

1 Introduction

Both in production and in service sectors, *project management* is a discipline of particular interest. Project-based organization and work is encountered within a very wide variety of applications: research and development (R&D), software development, construction, public infrastructure, process re-engineering, maintenance operations, ... A *project* itself can be informally defined as a unique undertaking, consisting of a set of precedence-related activities that have to be executed using diverse and mostly limited company resources. Project management deals with the selection and initiation of projects, as well as with their operation and control. *Project scheduling*, as a part of project management, is aimed at deciding when in time to start (and finish) which activities, and at the allocation of scarce resources to the project activities.

Unfortunately, project parameters such as activity durations and resource requirements are seldom precisely known and usually subject to estimation errors. Uncertainty is the prime cause of incomplete and unreliable data. This uncertainty can originate from a great number of potential sources. As some of the most frequently encountered causes, we can cite activities that take more or less time than originally estimated, machines breakdowns, materials that arrive behind schedule, worker absenteeism, delays due to bad weather... Although the sources of variability in the project environment are manifold, the main scheduling objectives are mostly related to the activities' starting (or ending) times, with the project makespan being the single most studied objective, next to other ones such as weighted earliness-tardiness and net present value of the project. This justifies a restriction to the study of uncertainty in processing times only, although many different sources may be at the basis of this variability.

Uncertainty in key project parameters is usually modeled in terms of probability theory [31]. Malcolm et al. [44] seem to have been the first in 1959 to recognize that randomness in the duration of a project's individual activities can be modeled by a stochastic variable. Subsequently, a large number of stochastic models for evaluating project duration have been developed, see for instance [1, 26, 37, 43]. All these studies neglect (renewable) resource constraints and assume that proper resource allocation decisions have already been made at a higher decision level. As coherently described by Stork [52], an important new aspect comes into play on moving from the deterministic to the stochastic case: what is a solution? A solution should define for each possible 'event' that occurs during the execution of the project an appropriate 'action', typically the start of new activities. To make such decisions, one may want to exploit the information given by the current state of the project. One schedule does not contain enough information to make decisions in all possible execution scenarios of the project. Stork (in line with Igelmund and Radermacher [30], among others) uses the term 'policy' to refer to a suitable set of decision rules that constitutes a solution. In the absence of resource constraints, the minimum-makespan objective requires no real scheduling effort: it is a dominant choice to start each activity as soon as its predecessors are completed. We formally define scheduling policies and related concepts in Section 3; most of the material on scheduling policies developed for stochastic scheduling can be transferred to robust optimization without major alterations.

Decision theory distinguishes between *risk*, *uncertainty* and *ignorance*. In a risk situation, the distribution of the outcomes under study is known with certainty. This is to be contrasted with 'unmeasurable' uncertainty, in a decision-theoretic context often simply termed 'uncertainty', in which it is not possible to attribute probabilities to the possible outcomes of a decision [27, 32, 50]. The case where even the possible outcomes are not known is usually referred to as 'unawareness', 'ignorance' or 'incomplete state space'; Loch et al. [42] speak of 'unk unks' (unknown unknowns). Rosenhead et al. [50] note that "it may be possible to convert an uncertainty problem into a risk problem, for example by the subjective estimation of probabilities, and used appropriately this can be a valuable simplification. However, some aspects of the future are genuinely unknowable, even in the probability sense. To insert notional probabilities may make the decision maker more comfortable, but that is not necessarily the objective in tackling a decision problem."

In this article, we study the resource-constrained project scheduling problem (RCPSP) (see [24, 34, 48] for surveys). Our purpose is to propose models for this scheduling problem that are useful when there is considerable uncertainty in the activity durations, and when the decision maker does not have sufficient confidence in the subjective probabilities that can be attributed to the different duration scenarios. Our modeling techniques stem from *robust optimization*, which is a theoretical framework that enables the decision maker to produce solutions that will have a reasonably good objective value under any likely input data scenario [2, 36].

The contributions of this article are threefold: (1) we describe how robust optimization can be applied to project scheduling under uncertainty; (2) we develop a scenario-relaxation algorithm to solve the optimization problem at hand; and (3) based on the scenario-relaxation algorithm, we develop a heuristic procedure that produces better results than two benchmark heuristics for medium-sized instances.

The remainder of this paper is organized as follows. First, we survey the literature on decision making under uncertainty in Section 2, with a particular focus on the objectives that are examined in the remainder of this paper. Subsequently, we give a number of definitions and a detailed problem statement in Section 3. The evaluation of the adopted objective function is discussed in Section 4, followed by a description of an optimization routine (Section 5) and of a heuristic procedure (Section 6). The results of our computational experiments on larger datasets are presented in Section 7. Finally, a summary and some conclusions are provided in Section 8.

2 Decision making under uncertainty

French [27] describes four criteria for decision making under uncertainty, which for a minimization problem amount to (1) *minimax*: minimize the worst makespan realization that can occur, (2) *minimin*: minimize the best outcome that can occur, which is an optimistic approach, as opposed to the pessimistic minimax, (3) *minimax regret*: minimize the largest possible difference in makespan between the policy to be selected and the optimal makespan for a given realization, and (4) minimize the objective *in expectation.* Within the context of this paper, the objectives (1) and (2) can be solved via the classic RCPSP, since the duration realizations of the different activities will be assumed to be independent of each other. The most-studied objective for the so-called *stochastic RCPSP* [5, 12, 52], where a probability distribution is known for the duration scenarios, is to select a policy that minimizes the expected value (4) of the project makespan within a specific class of policies. In the context of this article, however, probability distributions are not available and so expected values cannot be computed.

Assavapokee et al. [6] state that, because of incomplete information about the joint probability distribution of the uncertain parameters in the problem, decision makers are often unable to search for decisions with the best long-run average performance. Instead, they look for *robust* decisions, which perform well across all possible input scenarios without attempting to assign a fixed probability distribution to any ambiguous parameter. Daniels and Kouvelis [21] motivate the choice of regret-based objectives as follows: "a decision maker may be rightfully concerned not only with how a schedule's performance varies with the actual realizations of the task parameters, but also with how actual performance compares with the optimal performance that could have been achieved if perfect information had been available prior to scheduling. Such comparisons against optimal performance focus the decision maker on opportunities to free short-term capacity by reducing uncertainty and efficiently utilizing resources through scheduling, ...". Comparable regret-based objectives have recently been examined for various combinatorial optimization problems [6–8, 10, 38, 45]. Scheduling with regret-based objectives is studied in references [21, 35, 36] in a machine environment, with two objective functions: the *absolute-deviation robust* scheduling problem and the *relative-deviation robust* scheduling problem. The underlying deterministic machine problems studied in these references are easy (solvable in polynomial time). Aissi et al. [2] also point out that they prefer to study robust versions only of problems that are solvable in polynomial or pseudo-polynomial time, in the hope that they could preserve the complexity. The RCPSP, however, is strongly NP-hard [19].

Other approaches to robust optimization can be found in the literature; we briefly discuss some of these in the following lines. An extensive survey is given by Nikulin [49]. Ben-Tal and Nemirovski [13–15] find robust solutions to convex optimization problems with data uncertainty, when the data are drawn from ellipsoids; they produce solutions such that the constraints are respected whatever the realization of the data. A practical drawback of this approach is that it leads to non-linear, although convex, models, which are computationally rather demanding. Bertsimas and Sim [16, 17] propose an approach to address data uncertainty for discrete optimization and network-flow problems that allows the degree of conservatism of the solution to be controlled: protection is provided for the case where only a pre-specified number of the input coefficients changes from its base value, which allows to reduce the 'price of robustness' when the protection required is not too high. Finally, Mulvey et al. [46] present an approach that integrates goal-programming formulations with a scenario-based description of the problem data; they distinguish between solutions that remain close to optimal and those that remain 'almost feasible' and use the terms 'solution robust' and 'model robust', respectively.

3 Definitions and problem statement

In the following subsections, we provide some general definitions (Section 3.1), we outline the concept of scheduling policies (Section 3.2) and we give a detailed problem statement (Section 3.3). Section 3.4 discusses the difficulty of the evaluation of the objective function, Section 3.5 describes a dominance result and Section 3.6 contains our findings for an example project.

3.1 Project scheduling

We examine the scheduling of a single project. The project consists of a set $V = \{0, 1, \ldots, n+1\}$ of activities that need to be performed. We associate with each activity $i \in V$ a set $P_i \subset \mathbb{R}_+$ containing the possible realizations of the duration of activity i (with $\mathbb{R}_+$ the set of non-negative reals). This set P_i can be a discrete set $\{p_{i1}, p_{i2}, p_{i3}, \ldots, p_{i|P_i|}\}$ or an interval $[p_i^{\min}; p_i^{\max}]$; in the first case, we also write $p_i^{\min} \equiv \min_{P_i} p_{ik}$ and $p_i^{\max} \equiv \max_{P_i} p_{ik}$. The (dummy) activities 0 and $n+1$ have zero duration (meaning that $P_0 = P_{n+1} = \{0\}$). We use a lowercase vector $\mathbf{p} = (p_0, p_1, \ldots, p_{n+1})$ with $p_i \in P_i$ for all $i \in V$, to represent one particular *scenario* of the durations (also called *sample* or *realization*). The set containing all scenarios is denoted by $\mathcal{P} = P_0 \times P_1 \times \ldots \times P_{n+1}$: the possible durations for one activity are not dependent on the values chosen for the other activities.

When each $|P_i| = 1$, we are in the case of the deterministic RCPSP. Since each duration is a constant in this case, we also use the vector notation $\mathbf{p} = (p_0, \ldots, p_{n+1})$

for the durations. A solution to the RCPSP is a schedule $\mathbf{s}$, i.e., an $(n + 2)$-vector of starting times $(s_0, s_1, \dots, s_{n+1})$ with $s_i \geq 0$ for all $i \in V$. In most projects, some of the activities can only be started once other activities are completed. Such precedence relationships between the activities are represented by a binary relation $E \subset V \times V$. We assume that E is a (strict) partial order on V, i.e. an irreflexive and transitive relation. The activities 0 and $(n + 1)$ represent the start and the end of the project, respectively, meaning that $\forall i \in V \setminus \{0\} : (0, i) \in E$, and $\forall i \in V \setminus \{(n + 1)\} : (i, n + 1) \in E$, or in other words, 0 and $(n + 1)$ are predecessor, respectively successor, of all other activities. A so-called *precedence graph* $G(V, E)$ is inferred, where the nodes correspond to activities and arcs represent precedence relations. For a binary relation A on V, we let $T(A)$ denote its *transitive closure*, defined as the minimal transitive relation on V that contains A. Since E is transitive and irreflexive, G does not contain a cycle, and all precedence networks $G(V, A)$ with the same transitive closure $G(V, T(A))$ represent the same scheduling instance. The schedule $\mathbf{s}$ is said to be *precedence feasible* if $s_i + p_i \leq s_j$ for all $(i, j) \in E$. Without loss of generality, we usually set $s_0 = 0$.

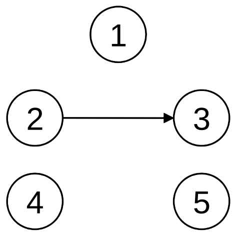
The project activities are to be scheduled on a set R of renewable resource types with availability b_k for each $k \in R$ (e.g., groups of equivalent workers or machines). Each activity $i \in V$ occupies a fixed number $b_{ik} \in \mathbb{N}$ units of each resource type k during its execution. The activities 0 and $n + 1$ do not use resources: $b_{0k} = b_{n+1,k} = 0$ for all $k \in R$. A schedule $\mathbf{s}$ is said to be *resource feasible* if, at any time t and for each resource type $k \in R$, it holds that $\sum_{i \in \mathcal{A}(\mathbf{s},t)} b_{ik} \leq b_k$, where the *active set* $\mathcal{A}(\mathbf{s}, t) = \{i \in V | s_i \leq t < s_i + p_i\}$ contains the activities in $V \setminus \{0, n + 1\}$ that are in progress at time t. The objective of the RCPSP is to find a precedence-feasible and resource-feasible schedule $\mathbf{s}$ that minimizes the project makespan s_{n+1}.

In this article, we examine the following problem: at the start of the project, the decision maker does not know which activity duration scenario will occur, and yet a number of sequencing decisions need to be made already (at least, he/she needs to decide which activities to release for execution at the start of the project horizon). We assume that an activity's duration realization is known only when the activity finishes (although this may implicitly be discovered earlier in the discrete case, namely as soon as the last-but-one scenario is exceeded). Sequencing decisions take the form of scheduling policies, which are the subject of the next subsection.

Figure 1(a) represents a precedence network for a small project with $n = 5$ non-dummy activities, so $V = \{0, 1, \dots, 6\}$ (the dummy nodes 0 and 6 are omitted for brevity). In our example project, the resource availability of a single resource type ($|R| = 1$) is $b_1 = 3$ units. All remaining data are provided in Figure 1(b). The schedule graphically represented in Figure 2 is a feasible schedule for this project when the activity durations are the components of the vector $\mathbf{p}^1 = (0, 2, 2, 2, 1, 2, 0)$.

3.2 Scheduling policies

The execution of a project with uncertain activity durations is a dynamic decision process. A solution is a *policy*, which defines *actions* at *decision times*. Decision times are typically the start of the project and the completion times of activities. An action can entail the start of a set of activities that is both precedence feasible and resource feasible. A schedule is thus constructed gradually through time. A decision at time t can only use information



(a) Project network

i	P_i	b_{i1}
0	$\{0\}$	0
1	$\{1,2,8\}$	1
2	$\{2,3\}$	2
3	$\{2\}$	1
4	$\{1,3\}$	2
5	$\{1,2\}$	1
6	$\{0\}$	0

(b) Other activity data

Figure 1: Example project network and activity data.

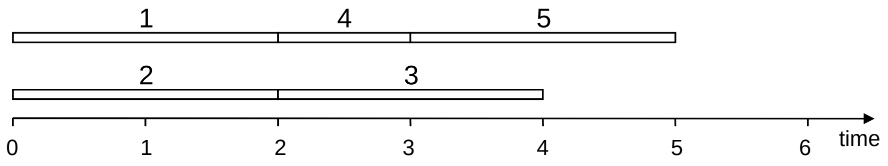


Figure 2: A feasible schedule for the example project under scenario $\mathbf{p}^1$.

that has become available before or at time t; this requirement is often referred to as the *non-anticipativity constraint.* As soon as all activities are completed, the activity durations are known, yielding a realization $\mathbf{p}$.

A set of activities $F \subset V$ is a *forbidden set* of a precedence relation A if it is an anti-chain of A (a stable set in graph $G(V, A)$) and if $\sum_{i \in F} b_{ik} > b_k$ for at least one $k \in R$: these sets can give rise to resource conflicts during project execution. A subset-minimal forbidden set is called a *minimal forbidden set* or *mfs* (see for instance [53]). The set of mfss for precedence relation A is written as $\mathcal{F}(A)$. For the example project presented in the previous subsection, we have $E = \{(2,3)\}$ and $\mathcal{F}(E) = \{\{1,2,5\},\{1,3,4\},\{2,4\},\{3,4,5\}\}$. Several scheduling policies for projects with stochastic activity durations were presented by Igelmund and Radermacher [30] based on the concept of forbidden sets. In this article, we study the set of earliest-start policies (ES-policies), which can be applied also when the probability distributions are not known. An ES-policy is characterized by a set of activity pairs $X \subset (V \times V) \setminus E$, such that for the extended set of activity pairs $E \cup X$ it holds that $\mathcal{F}(T(E \cup X)) = \emptyset$. The implication is that we can ignore the resource constraints if we respect the precedence constraints corresponding with $E \cup X$; in line with Balas [11], we call X a *selection.* The policy is feasible if $G(V, E \cup X)$ is still acyclic. A selection X of activity pairs that leads to a feasible ES-policy is called a *sufficient* set or *sufficient* selection.

An ES-policy parameterized by a sufficient selection X can be interpreted [30, 52] as a function $\mathbb{R}_+^{n+2} \mapsto \mathbb{R}_+^{n+2} : \mathbf{p} \mapsto \mathbf{s}(X, \mathbf{p})$ that maps given samples $\mathbf{p}$ of activity durations to feasible schedules $\mathbf{s}$. Let $G(V, E \cup X, \mathbf{p})$ denote the weighted graph where each arc $(j,k) \in E \cup X$ is valued by p_j. The starting time $s_i(X, \mathbf{p})$ is the length of a longest path from 0 to i in $G(V, E \cup X, \mathbf{p})$, which can be determined recursively (via standard longest-path calculations in acyclic graphs). The optimal makespan $s^*_{n+1}(\mathbf{p})$ for the RCPSP with durations $\mathbf{p}$ equals

$$s^*_{n+1}(\mathbf{p}) = \min_{X \in \mathcal{X}} s_{n+1}(X, \mathbf{p}), \tag{1}$$

where $\mathcal{X}$ is the set containing all sufficient selections. For known durations, this model is an extension of the disjunctive-graph representation of the classical job-shop scheduling problem [51], and has been known for quite some time already (see Balas [11], for instance).

In what follows, will use transshipment networks that represent the flow of resource units between activities; these networks are subsequently referred to as *(resource) flow networks*. Such networks have recently been proposed by various sources [4, 20, 41, 47, 48]. In this article, the word *flow* usually refers to a resource flow, unless noted otherwise. A flow f assigns to each triple $(i, j, k) \in V \times V \times R$ a value $f(i, j, k) \equiv f_{ijk} \in \mathbb{N}$, representing the number of resource units of type k that are transferred from the end of activity i to the start of activity j. These values must satisfy the following constraints, which are flow-conservation constraints as well as lower and upper bounds on the flow through intermediate nodes (not the start or end node):

$$\sum_{j \in V, j \neq i} f_{jik} = \sum_{j \in V, j \neq i} f_{ijk} = b_{ik}, \qquad \forall i \in V \setminus \{0, n+1\}, \forall k \in R.$$

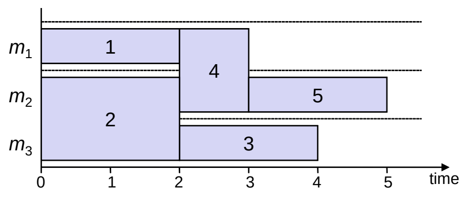
For each resource type $k \in R$, b_k resource units are sent into the network from the start node and collected at the end node:

$$\sum_{j \in V, j \neq 0} f_{0jk} = \sum_{j \in V, j \neq (n+1)} f_{j,n+1,k} = b_k, \qquad \forall k \in R.$$

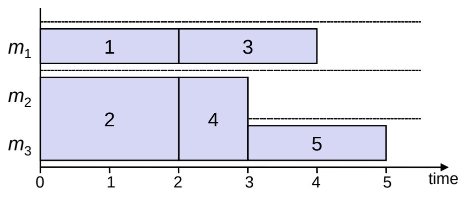
We are most interested in the flow-carrying arcs that are not in E, which do not coincide with technological precedence constraints; these are gathered in the set $C(f) = \{(i, j) \in V \times V : f(i, j, k) > 0 \text{ for at least one } k \in R\} \setminus E$. A flow f entails a detailed resource allocation decision for the individual units of each resource type, and induces additional precedence constraints via the elements of $C(f)$ under the condition of invariant resource allocation (see Bowers [20] for a discussion). We say that a flow f is *feasible* when $G(N, E \cup C(f))$ is acyclic, in which case the project can be implemented with the resource-allocation decisions inherent in f.

It is obvious that for a feasible flow f, $X = C(f)$ is a sufficient set; conversely, if the selection X defines a feasible ES-policy, then a feasible flow f exists with $E \cup C(f) \subseteq T(E \cup X)$. A further discussion of the equivalence between ES-policies and resource flows can be found in [39–41].

For the example project, the schedule depicted in Figure 2 does not provide information on the detailed allocation of the activities to the individual resources. Two possible allocations corresponding with the same schedule are depicted in Figures 3(a) and 3(b), where each horizontal band corresponds with a resource unit (e.g., one machine); the resource units are denoted by m_i $(i = 1, 2, 3)$. The resource flow networks corresponding with Figures 3(a) and 3(b) are depicted in Figures 4(a) and 4(b). The dummy activities 0 and 6 function as source and sink for the three resource units of the single resource type: the three units are dispatched into the network from activity 0 and gathered at node 6. Obviously, if more than one resource type is considered ($|R| > 1$), there will be a separate flow network for each resource type.

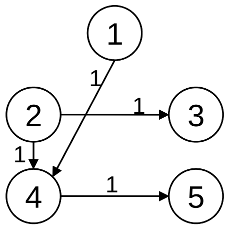


(a) Allocation 1

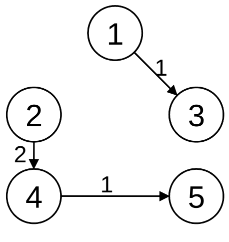


(b) Allocation 2

Figure 3: Two possible resource allocations for the example project; the durations correspond with scenario $\mathbf{p}^1$.



(a) Flow f_1



(b) Flow f_2

Figure 4: Flow networks corresponding with the resource allocations in Figures 3(a) and 3(b). Flow quantities are indicated next to each arc.

In the flow networks, some resource units are transported between activities that are not originally precedence-related (e.g., from activity 4 to 1 in case of f_1). If we decide to maintain the same resource allocation throughout the execution of the project then arcs such as $(4, 1)$ in the flow network induce additional 'hard' precedence constraints. In fact, once a decision has been made regarding the allocation of resources and as long as all (original and extra) precedence constraints are respected, we can disregard resource constraints altogether and still produce a resource-feasible schedule. The schedule in Figure 2, for instance, is the result of starting all activities as early as possible subject to the original precedence constraints augmented with the extra arcs from either Figure 4(a) or 4(b).

3.3 Problem statement

In this paper, we examine the minimax absolute-regret robust resource-constrained project scheduling problem or AR-RCPSP. The objective of the AR-RCPSP is to find an ES-policy that minimizes the maximum absolute regret over all scenarios. The absolute regret $\rho(X, \mathbf{p})$ for a sufficient selection X and duration vector $\mathbf{p}$ is the difference between the makespan $s_{n+1}(X, \mathbf{p})$ obtained by selection X and the optimal makespan $s^*_{n+1}(\mathbf{p})$ for $\mathbf{p}$, or

$$\begin{aligned}
\rho(X, \mathbf{p}) &= s_{n+1}(X, \mathbf{p}) - s^*_{n+1}(\mathbf{p}) \\
&= s_{n+1}(X, \mathbf{p}) - \min_{Y \in \mathcal{X}} s_{n+1}(Y, \mathbf{p}) \\
&= \max_{Y \in \mathcal{X}} \{ s_{n+1}(X, \mathbf{p}) - s_{n+1}(Y, \mathbf{p}) \} .
\end{aligned}$$

If we define the regret of a policy X relative to a policy Y as $\rho(X, Y, \mathbf{p}) = s_{n+1}(X, \mathbf{p}) - s_{n+1}(Y, \mathbf{p})$ then $\rho(X, \mathbf{p}) = \max_{Y \in \mathcal{X}} \rho(X, Y, \mathbf{p})$.

The maximum regret $\rho^{\max}(X)$ for a given sufficient selection X is

$$\begin{aligned}\rho^{\max}(X) &= \max_{\mathbf{p} \in \mathcal{P}} \rho(X, \mathbf{p}) \\ &= \max_{\mathbf{p} \in \mathcal{P}, Y \in \mathcal{X}} \{s_{n+1}(X, \mathbf{p}) - s_{n+1}(Y, \mathbf{p})\} = \max_{\mathbf{p} \in \mathcal{P}, Y \in \mathcal{X}} \rho(X, Y, \mathbf{p}).\end{aligned}$$

The optimization problem that we wish to solve can now be stated as follows:

$$(\textit{AR-RCPSP}) \qquad \rho^* = \min_{X \in \mathcal{X}} \rho^{\max}(X) = \min_{X \in \mathcal{X}} \max_{\mathbf{p} \in \mathcal{P}} \rho(X, \mathbf{p}) = \min_{X \in \mathcal{X}} \max_{\mathbf{p} \in \mathcal{P}, Y \in \mathcal{X}} \rho(X, Y, \mathbf{p}).$$

A problem closely related to AR-RCPSP is the minimax relative-regret robust resource-constrained project scheduling problem RR-RCPSP. For given X and $\mathbf{p}$, the relative regret $\tilde{\rho}(X, \mathbf{p})$ is given by:

$$\tilde{\rho}(X, \mathbf{p}) = \frac{s_{n+1}(X, \mathbf{p}) - s^*_{n+1}(\mathbf{p})}{s^*_{n+1}(\mathbf{p})} = \max_{Y \in \mathcal{X}} \frac{s_{n+1}(X, \mathbf{p})}{s_{n+1}(Y, \mathbf{p})} - 1 = \max_{Y \in \mathcal{X}} \tilde{\rho}(X, Y, \mathbf{p}),$$

where the last equality serves as a definition for $\tilde{\rho}(\cdot, \cdot, \cdot)$. The maximum relative regret can be written as follows:

$$\tilde{\rho}^{\max}(X) = \max_{\mathbf{p} \in \mathcal{P}, Y \in \mathcal{X}} \tilde{\rho}(X, Y, \mathbf{p}).$$

The RR-RCPSP then amounts to the following problem:

$$(\textit{RR-RCPSP}) \qquad \tilde{\rho}^* = \min_{X \in \mathcal{X}} \tilde{\rho}^{\max}(X) = \min_{X \in \mathcal{X}} \max_{\mathbf{p} \in \mathcal{P}} \tilde{\rho}(X, \mathbf{p}).$$

3.4 Objective-function evaluation

The RCPSP, which has known durations $\mathbf{p}$, is strongly NP-hard [19], and RCPSP reduces to the evaluation of the regret for a known selection X and duration vector $\mathbf{p}$: $\rho(X, \mathbf{p})$ is the difference between $s_{n+1}(X, \mathbf{p})$ and $s^*_{n+1}(\mathbf{p})$, where the first term can be obtained by a longest-path computation in $G(V, E \cup X, \mathbf{p})$ and the second term is the optimal solution to the RCPSP instance. Consequently, once we know $\rho(X, \mathbf{p})$, we also know $s^*_{n+1}(\mathbf{p})$. Hence, since the RCPSP is NP-hard, computing $\rho(X, \mathbf{p})$ is also NP-hard. A similar reasoning shows the NP-hardness of evaluating the relative regret.

Since computing the regret for a fixed ES-policy and duration vector is itself NP-hard, the computation of the maximum regret is not easy either. More precisely, evaluating the maximum absolute regret $\rho^{\max}(X)$ as well as the maximum relative $\tilde{\rho}^{\max}(X)$ is NP-hard for a given ES-policy defined by X: if the set of possible durations P_i is a singleton for each activity $i \in V$ then $|\mathcal{P}| = 1$ and the evaluation of $\rho^{\max}(X)$, respectively $\tilde{\rho}^{\max}(X)$, is equivalent to the evaluation of $\rho(X, \mathbf{p}^*)$, respectively $\tilde{\rho}(X, \mathbf{p}^*)$, where $\mathcal{P} = \{\mathbf{p}^*\}$. In project scheduling, when activity durations are decision variables, one deals with a so-called *multi-mode* scheduling problem. The problem of evaluating the maximum regret of a given ES-policy amounts to a multi-mode resource-constrained project scheduling problem.

By similar arguments, we also see that the computation of ρ^* and $\tilde{\rho}^*$ is hard: when $|\mathcal{P}| = 1$, minimizing the maximum regret amounts to finding a policy X with critical-path length of the extended network equal to the minimal RCPSP makespan, which is equivalent to solving the RCPSP.

3.5 Extreme duration scenarios

A duration scenario $\mathbf{p}$ is said to be *extreme* if $p_i = p_i^{\min}$ or $p_i = p_i^{\max}$ for all $i \in V$. According to [9], for the category of subset-type combinatorial optimization problems, the maximum regret can always be attained at an extreme scenario, and this for both the absolute and the relative maximum regret. In this section, we show that this is also the case for $\rho^{\max}(X)$ but not for $\tilde{\rho}^{\max}(X)$.

Theorem 1. *There is always an extreme duration scenario in which the maximum absolute regret of an ES-policy X is reached.*

Proof: For a duration vector $\mathbf{p}$, let $E(\mathbf{p})$ denote the set of activities with a duration that is strictly between its extreme values, so $E(\mathbf{p}) = \{i \in V | p_i^{\min} < p_i < p_i^{\max}\}$. We let $\mathbf{p}^*$ and Y^* represent a duration scenario and an ES-policy that achieve the maximum regret, i.e.

$$\rho(X, Y^*, \mathbf{p}^*) = s_{n+1}(X, \mathbf{p}^*) - s_{n+1}(Y^*, \mathbf{p}^*) = \rho^{\max}(X).$$

Suppose that $\mathbf{p}^*$ is not an extreme scenario (i.e., $|E(\mathbf{p}^*)| \geq 1$) and that $|E(\mathbf{p}^*)|$ is minimal. We choose an activity $j \in E(\mathbf{p}^*)$, so $p_j^{\min} < p_j^* < p_j^{\max}$. Recall that $s_{n+1}(X, \mathbf{p}^*)$ is equal to the length of $L_{X,\mathbf{p}^*}$, which denotes the longest path in $G(V, E \cup X, \mathbf{p}^*)$, while $s_{n+1}(Y^*, \mathbf{p}^*)$ is the length of $L_{Y^*,\mathbf{p}^*}$, the longest path in $G(V, E \cup Y^*, \mathbf{p}^*)$. Changing the duration of j from p_j^* to a different value in P_j preserves the feasibility of ES-policies X and Y^*. Let Y' denote an ES-policy of minimal makespan for a modified duration vector $\mathbf{p}'$, we have necessarily

$$s_{n+1}(Y', \mathbf{p}') \leq s_{n+1}(Y^*, \mathbf{p}'). \tag{2}$$

The following four possibilities are mutually exclusive and jointly exhaustive:

1. if $j \in L_{X,\mathbf{p}^*}$ and $j \in L_{Y^*,\mathbf{p}^*}$, let $\mathbf{p}'$ such that $p'_i = p^*_i$ for $i \neq j$ and $p'_j = p_j^{\max}$. $L_{X,\mathbf{p}^*}$ and $L_{Y^*,\mathbf{p}^*}$ remain the longest paths in the respective graphs with new lengths $s_{n+1}(X, \mathbf{p}') = s_{n+1}(X, \mathbf{p}^*) + p_j^{\max} - p_j^*$ and $s_{n+1}(Y^*, \mathbf{p}') = s_{n+1}(Y^*, \mathbf{p}^*) + p_j^{\max} - p_j^*$.

2. if $j \notin L_{X,\mathbf{p}^*}$ and $j \notin L_{Y^*,\mathbf{p}^*}$, let $\mathbf{p}'$ such that $p'_i = p^*_i$ for $i \neq j$ and $p'_j = p_j^{\min}$. $L_{X,\mathbf{p}^*}$ and $L_{Y^*,\mathbf{p}^*}$ remain the longest paths with unchanged lengths $s_{n+1}(X, \mathbf{p}') = s_{n+1}(X, \mathbf{p}^*)$ and $s_{n+1}(Y^*, \mathbf{p}') = s_{n+1}(Y^*, \mathbf{p}^*)$.

3. if $j \in L_{X,\mathbf{p}^*}$ and $j \notin L_{Y^*,\mathbf{p}^*}$, let $\mathbf{p}'$ such that $p'_i = p^*_i$ for $i \neq j$ and $p'_j = p_j^{\max}$. This keeps $L_{X,\mathbf{p}^*}$ as the longest path in $G(V, E \cup X, \mathbf{p}')$ and its length is increased, so $s_{n+1}(X, \mathbf{p}') = s_{n+1}(X, \mathbf{p}^*) + p_j^{\max} - p_j^*$. If a path in $G(V, E \cup Y^*, \mathbf{p}')$ becomes longer than $s_{n+1}(Y^*, \mathbf{p}^*)$, its length increases by at most $p_j^{\max} - p_j^*$. Hence we have $s_{n+1}(Y^*, \mathbf{p}') \leq s_{n+1}(Y^*, \mathbf{p}^*) + p_j^{\max} - p_j^*$.

4. if $j \notin L_{X,\mathbf{p}^*}$ and $j \in L_{Y^*,\mathbf{p}^*}$, we let $\mathbf{p}'$ such that $p'_i = p^*_i$ for $i \neq j$ and $p'_j = p_j^{\min}$. This keeps $L_{X,\mathbf{p}^*}$ as the longest path in $G(V, E \cup X, \mathbf{p}')$ and its length is unchanged with $s_{n+1}(X, \mathbf{p}') = s_{n+1}(X, \mathbf{p}^*)$. The length of the longest path in $G(V, E \cup Y^*, \mathbf{p}')$ decreases (by at most $p_j^* - p_j^{\min}$ time units), so we have $s_{n+1}(Y^*, \mathbf{p}') \leq s_{n+1}(Y^*, \mathbf{p}*)$.

For each of the four listed cases together with expression (2), the new regret verifies $s_{n+1}(X, \mathbf{p}') - s_{n+1}(Y', \mathbf{p}') \geq s_{n+1}(X, \mathbf{p}^*) - s_{n+1}(Y^*, \mathbf{p}')$. Hence, the value of p_j can be changed to $p_j^{\min}$ or $p_j^{\max}$ without decreasing $\rho(X, Y^*, \mathbf{p}^*)$, which reduces $|E(\mathbf{p}^*)|$ and contradicts the hypothesis of minimality. In conclusion, $\mathbf{p}^*$ and Y^* always exist such that $E(\mathbf{p}^*) = \emptyset$. $\square$

Unfortunately, the foregoing property does not hold for the relative regret. Consider a two-activity example with $P_1 = \{2, 3, 6\}$ and $P_2 = \{1, 3, 5\}$, without precedence constraints and with a resource usage such that the two activities can be scheduled in parallel. For any of the nine scenarios, the optimal makespan is $\max\{p_1, p_2\}$. Suppose we want to evaluate the maximum regret of ES-policy $X_1 = \{(1, 2)\}$. For any scenario $\mathbf{p} = (p_1, p_2)$ (we omit the dummy activities, for brevity), we have $s_{n+1}(X_1, \mathbf{p}) = p_1 + p_2$. The relative regret of ES-policy X_1 for duration scenario $\mathbf{p}$ is equal to

$$\begin{aligned}\tilde{\rho}(X_1, \mathbf{p}) &= \frac{p_1 + p_2}{\max\{p_1, p_2\}} - 1 \\ &= \frac{\min\{p_1, p_2\}}{\max\{p_1, p_2\}}.\end{aligned}$$

It is easily verified that the unique maximizer of this value is the duration scenario $(3, 3)$, which is not an extreme scenario. For this reason as well as due to the non-linearity inherent in the relative regret, we will focus only on the absolute regret in the remainder of this article. Note that the absolute regret $\rho(X_1, \mathbf{p})$ of policy X_1 is equal to $p_1 + p_2 - \max\{p_1, p_2\} = \min\{p_1, p_2\}$, which is maximized by the extreme scenario with $p_1 = 6$ and $p_2 = 5$.

3.6 Example project

For the example project presented in Section 3.1, define $X_1 = C(f_1)$ and $X_2 = C(f_2)$, with f_1 and f_2 as described in Figure 4. The regret of X_1 is maximized for duration vector $\mathbf{p}^2 = (p_{01}, p_{13}, p_{21}, p_{31}, p_{42}, p_{52}, p_{71}) = (0, 8, 2, 2, 3, 2, 0)$, with an optimal selection for this scenario being $Y = \{(2, 5), (4, 2)\}$, which isolates activity 1 on a separate resource unit because this activity may have a high duration (namely 8); the regret in this case is equal to the highest possible duration of activities 4 and 5, which are successors of activity 1 according to X_1. We have $\rho^{\max}(X_1) = \rho(X_1, \mathbf{p}^2) = s_6(X_1, \mathbf{p}^2) - s_6(Y, \mathbf{p}^2) = 13 - 8 = 5$. Similarly, the maximum regret of X_2 equals 2, which is the duration of activity 3 (which has only one possible value).

The maximum regret $\rho^{\max}(\cdot)$ is minimized by both the policies Y and X_2 and equals 2 (so $\rho^* = 2$). A maximum-regret scenario for Y is $\mathbf{p}^3 = (0, 1, 3, 2, 3, 1, 0)$, with $s_6(Y, \mathbf{p}^3) = 8$ while $s_6(Z, \mathbf{p}^3) = 6$, where $Z = \{(1, 3), (2, 1), (2, 4), (5, 4)\}$. The value of $\rho^{\max}(Z)$, on the other hand, is 5.

4 Evaluation of the maximum regret of an ES-policy

The absolute maximum regret of an ES-policy X is given by:

$$\rho^{\max}(X) = \max_{\mathbf{p}\in\mathcal{P}, Y\in\mathcal{X}} \{s_{n+1}(X, \mathbf{p}) - s_{n+1}(Y, \mathbf{p})\} = \max_{\mathbf{p}\in\mathcal{P}} \left\{ \max_{c\in\mathcal{C}(X,\mathbf{p})} l(c) - \min_{Y\in\mathcal{X}} \max_{c\in\mathcal{C}(Y,\mathbf{p})} l(c) \right\},$$

where $\mathcal{C}(Z, \mathbf{p})$ denotes the set of paths from 0 to $(n + 1)$ in $G(V, E \cup Z, \mathbf{p})$ for a selection Z and $l(c)$ is the length of the path c in the appropriate graph. The determination of each of these longest-path lengths can be cast into a linear formulation. Since the path lengths appear in the objective function with a positive sign for graph $G(V, E \cup X, \mathbf{p})$

and with a negative sign for $G(V, E \cup Y, \mathbf{p})$, we opt for the 'event-oriented' formulation for $G(V, E \cup X, \mathbf{p})$ and for the 'flow-oriented' formulation for $G(V, E \cup Y, \mathbf{p})$; see Wiest and Levy [54] for more details. This leads to

$$\rho^{\max}(X) = \max_{\mathbf{p} \in \mathcal{P}} \left\{ \left(\sum_{(i,j) \in E \cup X} p_i \phi_{ij} \right) - S^*(\mathbf{p}) \right\}$$

subject to

$$\begin{aligned}
\sum_{(i,j) \in E \cup X} \phi_{ij} &= \sum_{(j,i) \in E \cup X} \phi_{ji} & & \forall i \in V \setminus \{0, n+1\} \\
\sum_{(0,j) \in E \cup X} \phi_{0j} &= \sum_{(j,n+1) \in E \cup X} \phi_{j,n+1} = 1 & & \\
\phi_{ij} &\geq 0 & & \forall (i,j) \in E \cup X \\
S^*(\mathbf{p}) &= \min_{Y \in \mathcal{X}} \quad S_{n+1} & & \\
& \quad \text{subject to} \quad S_j \geq S_i + p_i & & \forall (i,j) \in E \cup Y \\
& \quad \phantom{\text{subject to}} \quad S_0 = 0 & &
\end{aligned}$$

By replacing the longest-path lengths by their linear-programming expressions, the maximum regret $\rho^{\max}(X)$ of a given selection X is the optimal objective value of a bi-level mathematical program with, in our case, an RCPSP instance at the lower level. The variables ϕ_{ij} search for the longest path in $G(V, E \cup X)$ by routing a unit flow through the network. An integration of the two levels of optimization is easily achieved:

$$\rho^{\max}(X) = \max_{\mathbf{p} \in \mathcal{P}} \left\{ \left(\sum_{(i,j) \in E \cup X} p_i \phi_{ij} \right) - S_{n+1} \right\} \tag{3}$$

subject to

$$\begin{aligned}
\sum_{(i,j) \in E \cup X} \phi_{ij} &= \sum_{(j,i) \in E \cup X} \phi_{ji} & & \forall i \in V \setminus \{0, n+1\} & (4) \\
\sum_{(0,j) \in E \cup X} \phi_{0j} &= \sum_{(j,n+1) \in E \cup X} \phi_{j,n+1} = 1 & & & (5) \\
\phi_{ij} &\geq 0 & & \forall (i,j) \in E \cup X & (6) \\
S_j &\geq S_i + p_i & & \forall (i,j) \in E \cup Y & (7) \\
S_0 &= 0 & & & (8) \\
Y &\in \mathcal{X} & & & (9)
\end{aligned}$$

We neglect for a moment the variable duration vector $\mathbf{p}$ and focus only on the RCPSP formulation with variable Y (cfr. (1)). The RCPSP formulation can be linearized using a resource-flow formulation, which has the benefit that it contains only a polynomial number of constraints and that it does not require the explicit determination of all minimal forbidden sets [22]. In particular, we add the following constraints:

$$\sum_{j\in V, j\neq i} f_{jik} = \sum_{j\in V, j\neq i} f_{ijk} = b_{ik} \qquad \forall i \in V\setminus\{0,n+1\}, \forall k\in R \tag{10}$$

$$\sum_{j\in V, j\neq 0} f_{0jk} = \sum_{j\in V, j\neq (n+1)} f_{j,n+1,k} = b_k \qquad \forall k\in R \tag{11}$$

$$f_{ijk} \geq 0 \qquad \forall (i,j)\in V\times V, \forall k \in R \tag{12}$$

The sufficient selection Y in the optimization problem is replaced by the set $C(f)$, with f a flow satisfying the above constraints (10)–(12) as well as acyclicity of $G(V, E\cup C(f))$. We replace the constraints (7) and (9) by the following:

$$0 \leq f_{ijk} \leq My_{ij} \qquad \forall (i,j)\in V\times V, \forall k\in R \tag{13}$$

$$S_j \geq S_i + p_i - M(1-y_{ij}) \qquad \forall (i,j)\in V\times V, i\neq j \tag{14}$$

$$y_{ij} = 1 \qquad \forall (i,j)\in E \tag{15}$$

$$y_{ij}\in\{0,1\} \qquad \forall (i,j)\in V\times V \tag{16}$$

The constraint sets (13) and (14) both contain 'big-M'-type constraints. The large number M can be chosen more specifically for each particular value of the indices i, j and k, a convenient choice is $\min\{b_{ik}, b_{jk}\}$ in (13) for $i, j \notin \{0, n+1\}$, with replacement of b_{ik} by b_k in this min-expression for $i = 0, n+1$. In (14), M can be an upper bound on the project makespan with durations $\mathbf{p}^{\max}$. When combined with the constraint sets (13)–(16), acyclicity of $G(V, E\cup C(f))$ is not an issue when the activity durations are non-zero.

Reverting to the optimization over $\mathcal{P}$ in (3), we should remove the non-linearity in the first term of (3) caused by the multiplication of p_i and ϕ_{ij} in order to obtain a linear model. According to Theorem 1, we need only consider two values $p_i^{\min}$ and $p_i^{\max}$ for p_i, with $p_i^{\min} = p_i^{\max} = 0$ for $i = 0, n+1$ (these zero values can be substituted immediately). We introduce n binary variables a_i ($i = 1,\ldots,n$), where $a_i = 0$ means that duration $p_i^{\min}$ is selected for activity i, while $a_i = 1$ indicates that $p_i = p_i^{\max}$. In Equation (14), we replace p_i by $(1-a_i)p_i^{\min} + a_i p_i^{\max}$. The non-linear terms $p_i\phi_{ij}$ in (3) are replaced by $p_i^{\min}\phi_{ij}^{\min} + p_i^{\max}\phi_{ij}^{\max}$ and each occurrence of ϕ_{ij} in the constraints is replaced by $\phi_{ij}^{\min} + \phi_{ij}^{\max}$, in which $\phi_{ij}^{\min}$ fulfills the role of ϕ_{ij} when $a_i = 0$ and $\phi_{ij}^{\max}$ functions as ϕ_{ij} in the cases where $a_i = 1$. This is achieved by adding the following two equation sets to the formulation:

$$\sum_{(i,j)\in E\cup X} \phi_{ij}^{\max} \leq a_i \qquad \forall i\in V\setminus\{0,n+1\} \tag{17}$$

$$\sum_{(i,j)\in E\cup X} \phi_{ij}^{\min} \leq 1 - a_i \qquad \forall i\in V\setminus\{0,n+1\} \tag{18}$$

Summarizing the foregoing, we find that the determination of the maximum regret of an ES-policy X reduces to an instance of a multi-mode RCPSP with a composite objective function consisting in the maximization of the difference between the length of the longest path in $G(V, E\cup X)$ and the optimal makespan S_{n+1}. We call this formulation *integrated*, because it simultaneously finds an optimal duration vector (an optimal scenario) and an optimal selection for the scenario (Y in the above). The full integrated formulation is included in the Appendix.

As an alternative, we propose a *scenario-based* formulation, in which some intermediate results are to be computed beforehand. We have

$$\begin{aligned}
\rho^{\max}(X) &= \max_{\mathbf{p}\in\mathcal{P}} \left\{ s_{n+1}(X,\mathbf{p}) - s^*_{n+1}(\mathbf{p}) \right\} \\
&= \min \rho && (19)\\
&\quad \text{subject to} \\
&\quad \rho \geq s_{n+1}(X,\mathbf{p}) - s^*_{n+1}(\mathbf{p}) \qquad \forall \mathbf{p}\in\mathcal{P} && (20)
\end{aligned}$$

If the longest-path length $s_{n+1}(X,\mathbf{p})$ and the RCPSP solution $s^*_{n+1}(\mathbf{p})$ are known for each scenario $\mathbf{p}$ then (19)–(20) is a linear formulation.

5 Absolute minimax-regret optimization

In this section, we present a procedure for finding an optimal solution to the problem AR-RCPSP, which was defined in Section 3.3. The procedure is based on an extension of the scenario-based formulation for evaluation of a given policy (see Section 4). In principle, we could also extend the integrated formulation. This, however, would lead to a rather cumbersome model, and the main interest of the scenario-based solution procedure lies in its modifications that will lead to an effective heuristic for the AR-RCPSP, as will be set out in Section 6.

When we plug the scenario-based model (19)–(20) into the definition of the absolute-regret objective, we obtain the following bi-level formulation of AR-RCPSP:

$$\begin{aligned}
\rho^* &= \min_{X\in\mathcal{X}} \rho^{\max}(X) \\
&= \min_{X\in\mathcal{X}} \left\{ \begin{array}{l} \min \rho \\ \text{subject to} \\ \rho \geq s_{n+1}(X,\mathbf{p}) - s^*_{n+1}(\mathbf{p}) \qquad \forall \mathbf{p}\in\mathcal{P} \end{array} \right\}
\end{aligned}$$

Again, we can assume that $\mathcal{P}$ contains only the extreme duration scenarios. The two levels of optimization are easily integrated and we resort to an event-based formulation for the longest-path computations in the graph $G(V, E\cup X)$. Let $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^{|\mathcal{P}|}\}$. This leads to

$$\rho^* = \min \rho \tag{21}$$

subject to

$$\begin{aligned}
\rho &\geq S^h_{n+1} - s^*_{n+1}(\mathbf{p}^h) && h = 1,\ldots,|\mathcal{P}| && (22)\\
S^h_j &\geq S^h_i + p^h_i - M(1 - x_{ij}) && \forall (i,j)\in V\times V, i\neq j, h = 1,\ldots,|\mathcal{P}| && (23)\\
S^h_i &\geq 0 && \forall i \in V, h = 1,\ldots,|\mathcal{P}| && (24)
\end{aligned}$$

together with the following scenario-independent constraints, which are similar to (10)–

(13) and (15)–(16) (substituting x_{ij} for y_{ij}):

$$\sum_{j\in V, j\neq i} f_{jik} = \sum_{j\in V, j\neq i} f_{ijk} = b_{ik} \qquad \forall i \in V \setminus \{0, n+1\}, \forall k \in R \tag{25}$$

$$\sum_{j\in V, j\neq 0} f_{0jk} = \sum_{j\in V, j\neq (n+1)} f_{j,n+1,k} = b_k \qquad \forall k \in R \tag{26}$$

$$f_{ijk} \geq 0 \qquad \forall (i,j) \in V \times V, \forall k \in R \tag{27}$$

$$0 \leq f_{ijk} \leq M x_{ij} \qquad \forall (i,j) \in V \times V, \forall k \in R \tag{28}$$

$$x_{ij} = 1 \qquad \forall (i,j) \in E \tag{29}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in V \times V \tag{30}$$

In the worst case, we have $|\mathcal{P}| = 2^n$. Hence, the MILP (mixed-integer linear program) includes an exponential number of variables S_i^h and constraints (22)–(24). Furthermore, for each duration vector $\mathbf{p}^h$ the optimal RCPSP solution $s_{n+1}^*(\mathbf{p}^h)$ has to be computed. We therefore investigate the possibility of solving a relaxed version of the foregoing formulation by only incorporating the constraints corresponding with a subset $\hat{\mathcal{P}} \subset \mathcal{P}$, iteratively adding scenarios until it can be guaranteed that the solution obtained for $\hat{\mathcal{P}}$ has the same objective as the full model with $\mathcal{P}$. Following Assavapokee et al. [6], we will refer to this approach as *scenario relaxation*. We call the resulting MILP the *master problem*, by analogy with Benders' decomposition, with objective function value $\rho^*(\hat{\mathcal{P}})$ for set $\hat{\mathcal{P}}$. Clearly, $\rho^*(\hat{\mathcal{P}})$ is a lower bound of $\rho^* \equiv \rho^*(\mathcal{P})$. The variables $S_i^h = 0$ for scenarios $\mathbf{p}^h \in \mathcal{P} \setminus \hat{\mathcal{P}}$ can be removed from the model without any influence.

In [6], a scenario-relaxation method is proposed to solve a general absolute min-max regret optimization problem with two-stage variables. The first stage-variables are binary "choice" variables, corresponding to our f and x representing the ES-policy. The second-stage variables are continuous "recourse" variables, in our case the variables S^h. To overcome the implementation problems caused by an exponential number of constraints, Assavapokee et al. [6] propose a three-stage algorithm, based on the iterative solution of the model on a restricted scenario set. The first stage consists in solving the master problem with a restricted scenario set so as to obtain a lower bound and the corresponding values for the first-stage decision variables. For a general optimization problem, these values can be infeasible for some scenarios excluded from the scenario set. For this reason, the second stage consists in finding such 'infeasible' scenarios, which are added to the scenario set and the algorithm returns to the first stage. If no infeasibilities are found, the algorithm proceeds to the third stage, which aims to identify a scenario in $\mathcal{P}\setminus\hat{\mathcal{P}}$ achieving the largest regret for the candidate robust solution x and f. In this paper, each ES-policy produced by the master problem will be feasible for all scenarios: the feasibility of an ES-policy is independent of the activity durations. A comparable iterative solution approach for an inventory model has recently been examined by Bienstock and Özbay [18].

We propose the framework described by Algorithm 1, in which LB and UB constitute a lower and upper bound on ρ^*, which are stepwise tightened over the course of the algorithm. Since an extreme duration vector is generated at each iteration, the solution framework converges within at most 2^n iterations. The restricted set of scenarios is updated at each iteration; we consider $\hat{\mathcal{P}}_q$ at iteration q. At initialisation (Step 1), $|\hat{\mathcal{P}}_1| = 1$ (although any number is possible). Bienstock and Özbay [18] call the master

Algorithm 1 scenario relaxation for AR-RCPSP

```
1: (initialisation)
   Consider duration-vector set $\hat{\mathcal{P}}_1$ containing a single duration vector $\mathbf{p}^1$. Set $q = 1$, $LB = 0$ and $UB = +\infty$. Compute $s^*_{n+1}(\mathbf{p}^1)$.
2: (master problem)
   Solve the restricted master problem (21)–(30) to obtain $LB = \rho^*(\hat{\mathcal{P}}_q)$ and the corresponding ES-policy $X_q$. If $LB = UB$ then stop.
3: (maximum-regret computation)
   Evaluate the maximum regret $\rho^{\max}(X_q)$ of policy $X_q$ using the integrated formulation of Section 4 and obtain the corresponding worst-case duration vector $\mathbf{p}^{q+1}$ and the associated optimal RCPSP makespan $s^*_{n+1}(\mathbf{p}^{q+1})$. Set $UB = \min\{\rho^{\max}(X_q); UB\}$.
4: (optimality check)
   If $LB = UB$ then stop; else set $q = q + 1$, $\hat{\mathcal{P}}_q = \hat{\mathcal{P}}_{q-1} \cup \{\mathbf{p}^q\}$ and go to Step 2.
```

problem (Step 2) the 'decision maker's problem', where the decision maker makes a first-stage decision (the ES-selection) while accounting for only a subset of the scenarios, and Step 3 is the 'adversarial problem', in which the worst scenario is generated for the candidate solution from Step 2, to verify its objective function against the full set of scenarios. Put differently, Step 3 looks for a scenario $\mathbf{p}^h$ that is not currently in $\hat{\mathcal{P}}$ and for which constraint (22) does not hold.

Computationally, the multi-mode-like instances in Step 3 of Algorithm 1 turn out to be especially hard. We can slightly modify the procedure by noting that at Step 3, it is not necessary to solve the maximum-regret evaluation to optimality. Let z represent the objective function of the subproblem; the correct functioning of the algorithm only requires that a duration vector $\mathbf{p}^{q+1}$ be found such that $z \geq LB + 1$. If one such duration vector exists then it can be included in the master problem, otherwise LB is optimal. To that purpose, we replace the objective function in the integrated formulation of Section 4 by

$$b^*(X) = \min b$$

and we add the constraints

$$\begin{aligned} z + b &\geq LB + 1 \\ z &\leq \left(\sum_{(i,j) \in E \cup X} p_i^{\min} \phi_{ij}^{\min} + p_i^{\max} \phi_{ij}^{\max} \right) - S_{n+1} \\ z, b &\geq 0 \end{aligned}$$

The resulting model has a solution $b^*(X) = 0$ if and only if there exists $z \geq LB + 1$. The drawback of this approach is that for the value S^*_{n+1} corresponding to a duration vector $\mathbf{p}^*$ output by this new subproblem, there is no guarantee that it equals the optimal makespan. Consequently, we additionally need to solve a standard RCPSP instance to obtain $s^*_{n+1}(\mathbf{p}^*)$ (in Step 3) before adding $\mathbf{p}^*$ to $\hat{\mathcal{P}}$ (in Step 4). We refer to the resulting subproblem as scenario generation with *bounded contribution.*

Algorithm 2 heuristic framework for AR-RCPSP

```
1: (initialisation)
   Consider duration-vector set $\hat{\mathcal{P}}_1$ containing a single duration vector $\mathbf{p}^1$, set $q = 1$ and
   compute $\hat{s}^*_{n+1}(\mathbf{p}^1)$.
2: (generate new solution)
   Use $\hat{s}^*_{n+1}(\mathbf{p}^q)$ and $\hat{\mathcal{P}}_q$ to produce a new approximate solution (ES-policy) $X_q$.
3: (generate new duration vectors)
   Generate one or more new duration vectors $\mathbf{p}^{q+1}$ that represent scenarios under which
   policy $X_q$ performs badly, together with an RCPSP upper bound $\hat{s}^*_{n+1}(\mathbf{p}^{q+1})$.
4: (iterate)
   Set $q = q + 1$, $\hat{\mathcal{P}}_q = \hat{\mathcal{P}}_{q-1} \cup \{\mathbf{p}^q\}$ and go to Step 2.
```

6 A heuristic for AR-RCPSP

Our computational results (see Section 7) indicate that the execution of the standard scenario-relaxation procedure (Algorithm 1) until convergence may take an inordinate amount of time even for medium-sized instances. We will therefore proceed with the development of heuristic solution procedures in this section, with the framework provided by Algorithm 1 as a basis. A first obvious such heuristic is the variant of Algorithm 1 that is not run until the stopping criterion $LB = UB$ is met, but rather until LB and UB are 'reasonably close' to each other, for instance when $(UB - LB)/LB < \epsilon$, with for example $\epsilon = 5\%$.

However, even this truncated run of Algorithm 1 will sometimes require very high running times, mainly due to the computational effort needed for performing Steps 2 and 3. We therefore propose a different approach, still following the same overall algorithmic structure but with significant efficiency gains also in each execution of Step 2 and 3; the main steps are presented as Algorithm 2. The essential drawback is that we again abandon the guarantee of finding an optimal solution: a number of approximations are inserted throughout the procedure. In the general variant of the algorithm, $\hat{s}^*_{n+1}(\mathbf{p}^h)$ is a heuristic solution (upper bound) to the RCPSP instance with optimal objective value $s^*_{n+1}(\mathbf{p}^h)$.

Step 2 produces a new solution, which is hopefully better than the current best solution. In our implementation, for the current scenario (duration vector $\mathbf{p}^h$), we solve the deterministic RCPSP to optimality (so $\hat{s}^*_{n+1}(\mathbf{p}^q)$ is actually $s^*_{n+1}(\mathbf{p}^{q+1})$ in our computations). With this schedule, we associate an activity list L that orders the activities in non-decreasing starting times (subsequently referred to as 'associated list'). This list is then compared to the list L^* associated with the current best solution. If the two are identical, the algorithm returns to Step 3 to obtain a new scenario, otherwise we consider all intermediary lists obtained by modifying the list L^* step-by-step until L is obtained. To each intermediary list, we apply a serial schedule generation scheme [33] to find a new schedule, which then in turn is used to produce a new solution (ES-policy) via the algorithm of [3]. This solution will be used as current best solution if it performs better (based on its regret) than the latter on the scenarios already generated. Let $L^* = (L^*_1, L^*_2, \ldots, L^*_n)$ and $L = (L_1, L_2, \ldots, L_n)$; notice that $L^*_0 = L_0$ and $L^*_{n+1} = L_{n+1}$. The intermediary lists between L^* and L are generated as follows. Let i be the lowest index for which $L_i^* \neq L_i$. We consider the list L' obtained from L^* by moving the activity L_i from its current position in L^* to position i in L'. Hence, the activities between position i and the current position of L_i in L^* are shifted. Next, we set $L^* = L'$ and repeat the procedure until $L' = L$. This step constitutes a path-relinking procedure [28, 29], generating feasible policies on a neighborhood path from an optimal policy (in terms of makespan for a given scenario) to another one.

In Step 3, a new scenario is generated by running the integrated formulation for evaluation of the current ES-policy X (as described in the Appendix) but from which all the y_{ij}-variables are removed (apart from those corresponding to $(i, j) \in E$): we effectively solve the problem

$$\max_{\mathbf{p} \in \mathcal{P}} \{s_{n+1}(X, \mathbf{p}) - s_{n+1}(\emptyset, \mathbf{p})\},$$

which delivers an upper bound for the actual regret of policy X and so also an upper bound for the minimax regret. This upper-bound formulation is handed to a MIP solver, which yields a new scenario. If the addition of this scenario does not lead to a solution different from the current best solution, we have a *cycling phenomenon.* In that case, we identify a longest path in the graph $G = (V, E \cup X)$, where X is the current best solution, to generate a new scenario: the activities on the path are set at their maximum durations, all other activities receive the minimum duration. In case this scenario also leads to cycling, a new scenario is generated randomly.

When solving the example project described in Section 3.1 using the basic implementation of Algorithm 1, the optimal value of 2 is obtained after three iterations and a running time of 0.03s. Using the implementation with bounded contribution, four iterations are used and the running time is only 0.02s. Finally, Algorithm 2 finds an optimal solution after six iterations, but its running time is 0s.

7 Computational results

All algorithms have been coded in C using Visual Studio C++ 2005; all the experiments were run on a Dell Optiplex 760 personal computer with Pentium R processor with 3.16 GHz clock speed and 3.21 GB RAM, equipped with Windows XP. CPLEX 10.2 was used for solving the MIP instances. Below, we first provide some details on the generation of the datasets, then we discuss some implementation details, and subsequently we present the computational results. Throughout this section, computation time is referred to as *time* and is expressed in seconds.

7.1 Data generation

The algorithms are tested on randomly generated instances of AR-RCPSP with n non-dummy activities, for n = 10, 20 and 30. We use the software RanGen [25] to generate instances of the deterministic RCPSP. Using this software, we can choose different values for the number of activities n, the order strength (OS), the resource factor (RF) and the resource constrainedness (RC) (for more information on these parameters, see [25]). For our experiments, we have chosen three different values for OS and two values for RF and RC, as follows. OS takes its value in the set $\{0.4, 0.6, 0.8\}$, RF is chosen from $\{0.45, 0.9\}$

and RC is a value in $\{0.3, 0.6\}$. For each combination of n, OS, RF and RC, we randomly generate ten instances of the deterministic RCPSP. From each RCPSP instance, we create an instance of AR-RCPSP by randomly choosing for each activity i with processing time p_i an integer δ_i between zero and $p_i - 1$. In the AR-RCPSP instance, the lower (upper) bound on the processing time of activity i is $p_i - \delta_i$ (respectively $p_i + \delta_i$). In total, there are $3 \times 2 \times 2 \times 10 = 120$ instances for each value of n.

7.2 Implementation details

In this section, we describe some of the implementation details for the algorithms proposed in Sections 5 and 6. We also present two simple heuristics that will serve as benchmarks in Section 7.3 for evaluation of the performance of the algorithms on the generated instances.

7.2.1 Algorithm 1

The implementation of Algorithm 1 follows the pseudocode of Section 5 with the following details and adaptations: we start with $\hat{\mathcal{P}}_1$ containing only $\mathbf{p}^1 = \mathbf{p}^{\min}$, the scenario in which the processing time of each activity is minimal. We use the branch-and-bound algorithm developed in [23] to solve the resulting deterministic RCPSP. Subsequently, the first master problem is set up but instead of solving it, we use the algorithm proposed in [3] to find a feasible resource flow and hence an ES-policy. The master problem is solved using CPLEX starting from the second iteration. We wish to underline that this implementation was chosen after preliminary experiments with other variants, among which an implementation where the initial scenario is chosen randomly, a variant where we initially add the two extreme scenarios (minimum and maximum durations) and one that initially adds three scenarios (minimum and maximum durations, and the third scenario is selected randomly).

Since Algorithm 1 is an exact procedure, we have investigated the bottleneck of its CPU time on the set of 10-activity instances. The CPU time of this algorithm is mainly made up of the time spent solving the master and the time needed to evaluate a given policy; we study the contribution of each of these two computations to the overall CPU time. In Table 1, we report the average CPU time for the master problem, for the evaluation procedure and the total average running time. We also report the average number of iterations (itr.) and the number of instances solved to guaranteed optimality within a time limit of 30 minutes (opt.). Each reported value in the table is the average of ten values, except in the last column (opt.). "Algorithm 1 (I)" refers to the "basic" implementation while "Algorithm 1 (II)" is the implementation "with bounded contribution". In general, the second implementation outputs results that are slightly better than those produced by the first implementation. Further, the implementation with bounded contribution cannot optimally solve one instance while the implementation without bounded contribution fails to solve two instances to optimality. We observe, however, that for one group (OS = 0.6, RF = 0.9 and RC = 0.6) the average CPU time of the implementation with bounded contribution is larger than the standard implementation. In fact, for that group of ten instances, there are two instances for which the implementation with bounded contribution takes substantially more time than the implementation without bounded contribution.

parameters			Algorithm 1 (I)					Algorithm 1 (II)				
			time			itr.	opt.	time			itr.	opt.
OS	RF	RC	Master	Evaluation	Total			Master	Evaluation	Total		
0.4	0.45	0.3	0.03	0.04	0.07	2.7	10	0.02	0.02	0.04	2.7	10
		0.6	0.12	1.15	1.27	2.1	10	0.12	0.71	0.83	2.1	10
	0.9	0.3	11.30	15.27	26.57	9.0	10	10.23	9.15	19.38	15.40	10
		0.6	16.48	882.59	899.07	1.6	8	19.06	734.84	753.90	4.25	9
0.6	0.45	0.3	0.01	0.02	0.03	2.1	10	0.01	0.02	0.03	2.8	10
		0.6	0.06	0.12	0.18	2.1	10	0.06	0.07	0.13	2.8	10
	0.9	0.3	0.15	0.40	0.55	3.4	10	0.17	0.24	0.41	4.3	10
		0.6	0.20	11.17	11.37	1.2	10	0.22	14.77	14.99	1.7	10
0.8	0.45	0.3	0.01	0.02	0.03	1.2	10	0.01	0.00	0.01	1.2	10
		0.6	0.01	0.02	0.03	1.2	10	0.01	0.02	0.03	1.2	10
	0.9	0.3	0.03	0.05	0.08	1.8	10	0.03	0.04	0.07	1.8	10
		0.6	0.05	0.14	0.19	1	10	0.05	0.14	0.19	1	10

Table 1: Distribution of average CPU time of Algorithm 1.

As mentioned, among the 120 instances, there is only one (for variant II) or two (for I) that are not solved within the time limit (for these two instances, the optimal solution was actually found but a certificate of optimality could not be produced within the time limit). These two instances belong to the same group with OS = 0.4, RF = 0.9 and RC = 0.6, which also has the highest average CPU time. We also observe that very few iterations are usually needed to arrive at an optimal solution: the average is never higher than 16, and in most cases even below 4; the algorithm with bounded contribution generally uses more iterations than the basic variant. When we compare the running times for evaluation and for the master problem, the former come out considerably higher than the latter.

7.2.2 Algorithm 2

The implementation of Algorithm 2 also follows its pseudocode. We again start with $\hat{\mathcal{P}}_1 = \{\mathbf{p}^{\min}\}$. For this minimum scenario, the deterministic RCPSP is solved and an ES-policy is constructed following [3]; this solution is set as the current best solution and is optimal for the scenario set $\hat{\mathcal{P}}_1$. In the first iteration, we can therefore skip Step 2 and immediately go to Step 3.

The stopping criteria for the algorithm are a limit on the computing time (30 minutes), a maximum number of scenarios generated (100) and a maximum number of consecutive scenarios giving rise to cycling (10). This implementation was chosen after preliminary experiments with different values for the maximum number of scenarios generated and the maximum number of consecutive scenarios engendering cycling.

7.2.3 Two simple heuristics

We present two additional heuristics that will be used as benchmarks for our two main algorithms. The rationale behind the choice for these simple heuristics is the fact that it is extremely difficult to provide meaningful bounds for AR-RCPSP even for medium-sized instances. The first heuristic is referred to as Heuristic 1 and is described in pseudocode below. As before, the deterministic RCPSP is solved using the branch-and-bound algorithm developed in [23].

Heuristic 1

```
1: determine the average duration $p_i = \lfloor \frac{p_i^{min}+p_i^{max}}{2} \rfloor$ for each activity $i$
2: solve the corresponding deterministic RCPSP
3: impose a resource flow on this deterministic schedule with the algorithm of [3]
4: output the solution found
```

The second heuristic is named Heuristic 2 and is outlined below. The problem encountered in Step 2 is formulated as a MIP where the objective is to minimize the number of arcs in the transitive closure, which is solved using CPLEX. We refer to [39] for a motivation for this choice of objective function.

Heuristic 2

```
1: ignore the activity durations
2: find a solution (ES-policy) with a transitive closure having the minimum number of arcs
```

7.3 Computational experiments

Below, we present our computational results for instance sets with 10, 20 and 30 activities.

7.3.1 Comparison of the algorithms for 10-activity instances

Table 2 displays for every algorithm the average CPU time (time), the number of instances for which the algorithm finds an optimal solution (opt.) and the average gap (gap*) for the set of instances with ten non-dummy activities. We opt for variant II of Algorithm 1. In the table, each value reported in the columns time and gap* is the average of ten values. Since the optimal objective value can be zero, the usual 'gap' is not well defined. Therefore, throughout this section, we use gap* defined as follows. For an instance, let $f(H)$ be the value of the solution found by the considered algorithm and $f(opt)$ be the optimal objective value (for $n = 10$, this optimal value is found by Algorithm 1). Further, let δ be the average difference between minimum and maximum duration of activities, so $\delta = \frac{1}{n}\sum_{i=1}^{n}\left(p_i^{\max} - p_i^{\min}\right)$. We define gap* $= 100 \times \frac{f(H)-f(opt)}{\delta}$.

Table 2 shows that Algorithm 1 optimally solves all the instances; this, however, is sometimes coupled with a high average CPU time. Among the remaining algorithms, Algorithm 2 solves 82 instances out of 120 to optimality, while Heuristic 1 finds optimal solutions for 47 instances and Heuristic 2 provides optimal solutions for only 40 instances. Further, Algorithm 2 usually produces the smallest average gap* for this dataset. However, the average CPU time of Algorithm 2 is higher than Heuristic 2, which, in turn, is higher than Heuristic 1.

7.3.2 Comparison for 20-activity instances

In this section, we focus on the set of instances with 20 non-dummy activities. For these instances, attempts to provide optimal solutions by solving the mixed-integer formulation

parameters			Algorithm 1			Algorithm 2			Heuristic 1			Heuristic 2		
OS	RF	RC	time	gap*	opt.	time	gap*	opt.	time	gap*	opt.	time	gap*	opt.
0.4	0.45	0.3	0.04	0.00	10	2.03	17.78	5	0.00	26.09	2	0.02	75.98	2
		0.6	0.83	0.00	10	2.53	31.37	5	0.00	80.21	3	0.04	90.12	2
	0.9	0.3	19.38	0.00	10	3.88	44.87	4	0.00	59.71	3	0.02	62.35	3
		0.6	753.90	0.00	9	2.58	25.00	5	0.00	138.70	0	0.06	45.60	3
0.6	0.45	0.3	0.03	0.00	10	1.36	28.21	7	0.00	21.65	6	0.02	52.57	5
		0.6	0.13	0.00	10	2.44	25.00	7	0.00	30.93	4	0.04	38.56	3
	0.9	0.3	0.41	0.00	10	2.24	30.51	4	0.00	52.07	3	0.07	91.88	2
		0.6	14.99	0.00	10	1.86	11.90	8	0.00	20.41	6	0.44	33.15	4
0.8	0.45	0.3	0.01	0.00	10	0.21	0.00	10	0.00	7.32	7	0.01	94.59	4
		0.6	0.03	0.00	10	0.00	0.00	10	0.00	13.96	7	0.13	47.53	6
	0.9	0.3	0.07	0.00	10	1.22	17.50	7	0.00	75.02	0	0.46	199.15	0
		0.6	0.19	0.00	10	0.73	0.00	10	0.00	11.10	6	0.90	10.40	6

Table 2: Comparison for $n = 10$.

with CPLEX produced extremely poor results within the time limit of 30 minutes: when interrupting CPLEX after 30 minutes, the best lower bound is usually zero and the best upper bound is very large. Without an optimal value nor a good lower bound, gap* defined in the previous section is meaningless. Therefore, to compare the different algorithms we first report the average CPU time and the number of optimal solutions found in Table 3. In fact, each algorithm may have found more optimal solutions than what is reported in Table 3 because we count only the number of optimal solutions with objective value zero.

parameters			Algorithm 1 (I)		Algorithm 1 (II)		Algorithm 2		Heuristic 1		Heuristic 2	
OS	RF	RC	time	opt.	time	opt.	time	opt.	time	opt.	time	opt.
0.4	0.45	0.3	629.22	8	542.31	8	2.00	1	0.00	0	14.72	1
		0.6	2201.50	0	1947.81	1	3.22	0	0.00	0	1207.69	1
	0.9	0.3	2340.17	0	2247.11	0	3.59	0	0.00	0	1800.00	0
		0.6	1985.90	0	1830.47	1	5.18	0	0.00	0	1800.00	0
0.6	0.45	0.3	56.42	10	61.00	10	1.63	2	0.00	2	0.21	1
		0.6	1990.98	0	1559.89	0	2.57	1	0.00	1	36.24	0
	0.9	0.3	2208.07	0	1466.95	0	2.58	3	0.00	1	1153.71	0
		0.6	1800.10	0	2081.57	1	1.87	0	0.00	0	1800.16	0
0.8	0.45	0.3	1.27	10	0.68	10	0.84	5	0.00	1	0.06	3
		0.6	190.95	10	133.15	10	1.10	4	0.00	1	0.37	2
	0.9	0.3	183.94	10	177.56	10	1.41	2	0.00	0	0.56	1
		0.6	1799.99	4	1808.54	7	0.29	0	0.00	0	113.78	0

Table 3: Comparison for $n = 20$.

Variant I of Algorithm 1 provides guaranteed optimal solutions to 52 instances out of 120 within the time limit of 30 minutes; this number goes up to 58 for the bounded-contribution implementation. The average CPU time of this algorithm is very large for both implementations. We observe that for most unsolved instances, the time limit is reached before two iterations are fully completed. This confirms that solving both the master problem and the evaluation problem in an exact fashion is simply overly time-consuming. Algorithm 2 produces optimal solutions for 18 instances, with a maximum average CPU time less than six seconds. Heuristic 1 is the fastest algorithm but obtains optimal solutions for only six instances, while Heuristic 2 provides such solutions for nine instances but needs more time.

In order to further compare the quality of the outputs of the algorithms, we have performed a pair-wise comparison, the results of which are reported in Table 4. For a given instance of AR-RCPSP, let X and Y be the solutions output by two different algorithms. Using CPLEX, we compute the quantity

$$LB(X,Y) = \max_{p \in \mathcal{P}} \Big\{ s_{n+1}\big(X,p\big) - s_{n+1}\big(Y,p\big) \Big\},$$

which entails the maximum regret of the output of the first algorithm with respect to the output of the second algorithm over all the scenarios; $LB(X,Y)$ is a lower bound for $\rho^{\max}(X)$ for any feasible Y. The quantity $LB(Y,X)$ represents a similar comparison of the second algorithm with respect to the first one; $LB(X,Y)$ and $LB(Y,X)$ need not be the same. In Table 4, we denote Algorithm 1 (with bounded contribution) by A1, Algorithm 2 by A2, Heuristic 1 by H1 and Heuristic 2 by H2. Observe that any number in this table is the average of ten values. We conclude that Algorithm 1 achieves the best comparison for those instances where it regularly finds optimal solutions. Overall, Algorithm 2 tends to display the smallest difference with respect to the output of the other algorithms. Between Heuristic 1 and Heuristic 2, however, there is no clear domination of one over the other.

parameters			Algorithm 1 (A1)			Algorithm 2 (A2)			Heuristic 1 (H1)			Heuristic 2 (H2)		
OS	RF	RC	A2	H1	H2	A1	H1	H2	A1	A2	H2	A1	A2	H1
0.4	0.45	0.3	0.00	0.00	0.00	31.48	12.73	8.22	42.87	23.37	46.70	87.12	62.26	32.25
		0.6	49.29	46.31	30.66	33.12	22.70	12.10	42.38	28.49	38.12	49.75	121.16	73.56
	0.9	0.3	80.63	45.34	72.98	38.10	10.63	28.95	119.21	164.25	46.11	71.08	135.26	58.63
		0.6	89.45	22.39	66.87	4.56	0.20	0.00	92.08	178.24	47.32	21.70	166.96	65.45
0.6	0.45	0.3	0.00	0.00	0.00	36.21	26.12	47.87	76.36	46.85	0.00	55.18	65.14	23.90
		0.6	79.05	39.23	54.65	42.92	32.61	0.00	68.49	155.15	45.17	32.57	75.51	33.80
	0.9	0.3	77.81	64.52	39.06	42.65	78.93	96.35	69.84	131.16	37.54	49.01	124.32	62.64
		0.6	51.26	25.79	49.07	33.64	52.68	72.95	44.98	73.63	64.94	27.23	59.10	37.24
0.8	0.45	0.3	0.00	0.00	0.00	33.29	15.23	13.25	67.66	47.56	12.52	38.11	31.50	11.86
		0.6	0.00	0.00	0.00	75.42	23.56	36.45	92.35	45.68	25.23	74.00	40.27	39.60
	0.9	0.3	0.00	0.00	0.00	47.38	25.12	24.65	109.22	78.54	23.69	164.47	66.45	46.10
		0.6	18.04	8.59	13.37	68.00	39.24	27.45	99.65	64.52	45.63	139.80	64.00	31.50

Table 4: Pair-wise comparison for $n = 20$.

7.3.3 Comparison for 30-activity instances

We have performed comparisons for the set with 30 non-dummy activities similar to the previous sections. We do not, however, include the exact algorithm (Algorithm 1) because within the imposed time limit, this algorithm is unable to perform even a single iteration. In Table 5, we compare the average CPU time and the number of solutions with zero objective value found by each of the three remaining algorithms. Algorithm 2 obtains solutions with zero objective value for nine instances while Heuristic 1 and Heuristic 2 do so for only one instance. Again, it is important to note that this does not mean that these algorithms did not solve more instances until optimality, since we do not know the optimal objective value. From Table 5, we also observe that (somewhat logically) the average CPU time of each algorithm has increased compared to Table 3. The pair-wise comparison of the three algorithms is described in Table 6. For this dataset, Algorithm 2 displays the smallest values when compared to the other two heuristics.

parameters			Algorithm 2		Heuristic 1		Heuristic 2	
OS	RF	RC	time	opt.	time	opt.	time	opt.
0.4	0.45	0.3	4.99	2	0.00	0	0.27	0
		0.6	6.89	1	0.00	0	105.54	1
	0.9	0.3	25.79	0	0.00	0	703.85	0
		0.6	17.59	0	0.00	0	1800.17	0
0.6	0.45	0.3	21.15	1	0.00	1	976.51	0
		0.6	17.30	1	0.00	0	1800.06	0
	0.9	0.3	52.18	0	0.01	0	1800.03	0
		0.6	53.07	0	0.00	0	1800.04	0
0.8	0.45	0.3	14.74	2	0.01	0	1753.83	0
		0.6	11.62	1	0.01	0	1800.07	0
	0.9	0.3	31.81	1	1.19	0	1800.05	0
		0.6	32.46	0	0.00	0	1800.02	0

Table 5: Comparison for $n = 30$.

parameters			A2		H1		H2	
OS	RF	RC	H1	H2	A2	H2	A2	H1
0.4	0.45	0.3	36.90	41.65	150.30	57.24	239.00	77.62
		0.6	39.20	43.70	403.50	108.31	344.00	94.87
	0.9	0.3	15.50	18.80	381.80	163.44	310.10	124.64
		0.6	55.20	58.80	420.10	122.18	390.70	210.23
0.6	0.45	0.3	34.20	39.40	244.40	132.52	197.00	97.56
		0.6	39.80	54.90	332.00	142.10	304.60	56.78
	0.9	0.3	34.60	56.50	354.80	180.30	365.10	169.23
		0.6	73.70	86.90	467.30	156.34	444.80	274.59
0.8	0.45	0.3	18.50	28.10	206.60	93.24	169.30	73.52
		0.6	57.00	86.40	376.00	196.03	358.80	216.76
	0.9	0.3	59.70	90.50	406.70	215.73	400.80	91.08
		0.6	80.20	100.80	445.20	106.07	490.30	134.27

Table 6: Pair-wise comparison for $n = 30$.

8 Summary and conclusions

In practical project management, a project's parameters such as activity durations and resource requirements are seldom precisely known and usually subject to estimation errors. In this article we have proposed a robust optimization approach to project scheduling with uncertain activity durations, assuming that the decision maker cannot with confidence associate probabilities with possible activity durations. The resulting robust project scheduling problem that we have studied, has turned out to be exceptionally difficult, in that even exact objective-function evaluation is intractable and computationally overly demanding, even for medium-sized instances. We have developed and implemented a scenario-relaxation algorithm and a scenario-relaxation-based heuristic. The first algorithm produces optimal solutions but requires excessive running times even for medium-sized instances; the second algorithm produces high-quality solutions for medium-sized instances, which are significantly better than those produced by two benchmark heuristics – although the latter consume less CPU time.

Further research should be oriented towards the formulation and solution of more practical variants of the AR-RCPSP, for instance by considering an objective function that can be evaluated in polynomial time for a given scheduling policy, so that the corresponding decision problem is at least in NP. This could be the case for the minimization of the upper bound of the minimax regret of a policy X defined as $\max_{\mathbf{p}\in\mathcal{P}} \{s_{n+1}(X,\mathbf{p}) - s_{n+1}(\emptyset,\mathbf{p})\}$, the complexity status of which, to the best of our knowledge, is open.

Acknowledgments

This work was supported by research project G.0508.09 of the Belgian Research Foundation – Flanders (FWO) and by the French National Research Agency (ANR) project on Robustness and Cooperation in Scheduling "ROBOCOOP" ANR-08-BLAN-0331-01.

Appendix

The full integrated formulation for evaluation of the maximum regret of a policy X, which was presented in Section 4, is given below.

$$\rho^{\max}(X) = \quad \max \left\{ \left(\sum_{(i,j)\in E\cup X} p_i^{\min}\phi_{ij}^{\min} + p_i^{\max}\phi_{ij}^{\max} \right) - S_{n+1} \right\}$$

subject to

$$\begin{aligned}
\sum_{(i,j)\in E\cup X} \phi_{ij}^{\min} + \phi_{ij}^{\max} &= \sum_{(j,i)\in E\cup X} \phi_{ji}^{\min} + \phi_{ji}^{\max} && \forall i \in V \setminus \{0, n+1\} \\
\sum_{(0,j)\in E\cup X} \phi_{0j}^{\min} + \phi_{0j}^{\max} &= \sum_{(j,n+1)\in E\cup X} \phi_{j,n+1}^{\min} + \phi_{j,n+1}^{\max} = 1 \\
0 \le f_{ijk} &\le M y_{ij} && \forall (i,j) \in V \times V, \forall k \in R \\
S_j &\ge S_i + (1-a_i)p_i^{\min} + a_i p_i^{\max} - M(1-y_{ij}) && \forall (i,j) \in V \times V, i \ne j \\
\sum_{j\in V, j\ne i} f_{jik} &= \sum_{j\in V, j\ne i} f_{ijk} = b_{ik} && \forall i \in V \setminus \{0, n+1\}, \forall k \in R \\
\sum_{j\in V} f_{0jk} &= \sum_{j\in V} f_{j,n+1,k} = b_k && \forall k \in R \\
\sum_{(i,j)\in E\cup X} \phi_{ij}^{\max} &\le a_i && \forall i \in V \setminus \{0, n+1\} \\
\sum_{(i,j)\in E\cup X} \phi_{ij}^{\min} &\le 1 - a_i && \forall i \in V \setminus \{0, n+1\} \\
\phi_{ij}^{\min}, \phi_{ij}^{\max} &\ge 0 && \forall (i,j) \in E \cup X \\
y_{ij} &= 1 && \forall (i,j) \in E \\
y_{ij} &\in \{0,1\} && \forall (i,j) \in V \times V \\
S_0 &= 0 \\
f_{ijk} &\ge 0 && \forall (i,j) \in V \times V, \forall k \in R \\
a_i &\in \{0,1\} && \forall i \in V \\
a_0 &= a_{n+1} = 0
\end{aligned}$$

References

[1] V.G. Adlakha and V.G. Kulkarni. A classified bibliography of research on stochastic PERT networks: 1966-1987. *INFOR*, 27:272–296, 1989.

[2] H. Aissi, C. Bazgan, and D. Vanderpooten. Min-max and min-max regret versions of combinatorial optimization problems: a survey. *European Journal of Operational Research*, 197:427–438, 2009.

[3] C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[4] C. Artigues and F. Roubellat. A polynomial activity insertion algorithm in a multiresource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, 127:297–316, 2000.

[5] B. Ashtiani, R. Leus, and M.B. Aryanezhad. New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of preprocessing. *Journal of Scheduling*. To appear.

[6] T. Assavapokee, M.J. Realff, and J.C. Ammons. A new min-max regret robust optimization approach for interval data uncertainty. *Journal of Optimization Theory and Applications*, 137:297–316, 2008.

[7] T. Assavapokee, M.J. Realff, J.C. Ammons, and I.H. Hong. Scenario relaxation algorithm for finite scenario-based min-max regret and min-max relative regret robust optimization. *Computers & Operations Research*, 35:2093–2102, 2008.

[8] I. Averbakh. Minmax regret solutions for minimax optimization problems with uncertainty. *Operations Research Letters*, 27:57–65, 2000.

[9] I. Averbakh. Computing and minimizing the relative regret in combinatorial optimization with interval data. *Discrete Optimization*, 2:273–287, 2005.

[10] I. Averbakh and V. Lebedev. Interval data minmax regret network optimization problems. *Discrete Applied Mathematics*, 138:289–301, 2004.

[11] E. Balas. Project scheduling with resource constraints. In E.M.L. Beale, editor, *Applications of Mathematical Programming Techniques*, pages 187–200. The English Universities Press Ltd, 1971.

[12] F. Ballestín and R. Leus. Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management*, 18(4):459–474, 2009.

[13] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25:1–13, 1999.

[14] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88:411–424, 2000.

[15] A. Ben-Tal and A. Nemirovski. Robust optimization – methodology and applications. *Mathematical Programming*, 92:453–480, 2002.

[16] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98:49–71, 2003.

[17] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52:35–53, 2004.

[18] D. Bienstock and N. Özbay. Computing robust basestock levels. *Discrete Optimization*, 5:389–414, 2008.

[19] J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[20] J.A. Bowers. Criticality in resource constrained networks. *Journal of the Operational Research Society*, 46:80–91, 1995.

[21] R.L. Daniels and P. Kouvelis. Robust scheduling to hedge against processing time uncertainty in single-stage production. *Management Science*, 41:363–376, 1995.

[22] S. Demassey. Mathematical programming formulations and lower bounds. In C. Artigues, S. Demassey, and E. Néron, editors, *Resource-Constrained Project Scheduling. Models, Algorithms, Extensions and Applications*, pages 49–62. ISTE Ltd. and John Wiley and Sons, Inc., 2008.

[23] E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818, 1992.

[24] E. Demeulemeester and W. Herroelen. *Project Scheduling. A Research Handbook.* Kluwer Academic Publishers, 2002.

[25] E. Demeulemeester, M. Vanhoucke, and W. Herroelen. A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6:13–34, 2003.

[26] S.E. Elmaghraby. *Activity Networks: Project Planning and Control by Network Models.* Wiley, 1977.

[27] S. French. *Decision Theory. An Introduction to the Mathematics of Rationality.* Ellis Horwood Limited, 1988.

[28] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Boston, 1997.

[29] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[30] G. Igelmund and F.J. Radermacher. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13:1–28, 1983.

[31] T. Jørgensen. *Project scheduling as a stochastic dynamic decision problem.* PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, 1999.

[32] F.H. Knight. *Risk, Uncertainty, and Profit.* Houghton Mifflin, Boston, 1921.

[33] R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996.

[34] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *Omega*, 29:249–272, 2001.

[35] P. Kouvelis, R.L. Daniels, and G. Vairaktarakis. Robust scheduling of a two-machine flow shop with uncertain processing times. *IIE Transactions*, 32:421–432, 2000.

[36] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications.* Kluwer Academic Publishers, 1997.

[37] V.G. Kulkarni and V.G. Adlakha. Markov and Markov-regenerative PERT networks. *Operations Research*, 34:769–781, 1986.

[38] V. Lebedev and I. Averbakh. Complexity of minimizing the total flow time with interval data and minmax regret criterion. *Discrete Applied Mathematics*, 154:2167–2177, 2006.

[39] R. Leus. Resource allocation by means of project networks: complexity results. *Networks*, to appear.

[40] R. Leus. Resource allocation by means of project networks: dominance results. *Networks*, to appear.

[41] R. Leus and W. Herroelen. Stability and resource allocation in project planning. *IIE Transactions*, 36:667–682, 2004.

[42] C.H. Loch, A. De Meyer, and M.T. Pich. *A New Approach to Managing High Uncertainty and Risk in Projects.* Wiley, 2006.

[43] A. Ludwig, R.H. Möhring, and F. Stork. A computational study on bounding the makespan distribution in stochastic project networks. *Annals of Operations Research*, 102:49–64, 2001.

[44] D.G. Malcolm, J.M. Rosenbloom, C.E. Clark, and W. Fazar. Application of a technique for research and development program evaluation. *Operations Research*, 7:646–669, 1959.

[45] R. Montemanni. A mixed integer programming formulation for a single machine robust scheduling with interval data. *Journal of Mathematical Modelling and Algorithms*, 6:287–296, 2007.

[46] J.M. Mulvey, R.J. Vanderbei, and S.A. Zenios. Robust optimization of large-scale systems. *Operations Research*, 43:264–281, 1995.

[47] G. Naegler and S. Schoenherr. Resource allocation in a network model - the Leinet system. In R. Slowinski and J. Weglarz, editors, *Advances in project scheduling*, chapter II.8. Elsevier, 1989.

[48] K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources: Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions*. Springer-Verlag, 2003.

[49] Y. Nikulin. Robustness in combinatorial optimization and scheduling theory: an extended annotated bibliography. Technical Report 606, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Kiel, Germany, 2006.

[50] J. Rosenhead, M. Elton, and S.K. Gupta. Robustness and optimality as criteria for strategic decisions. *Operations Research Quarterly*, 23:413–431, 1972.

[51] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Technical Report Note D.S. no. 9 bis, SEMA, Paris, France, 1964.

[52] F. Stork. *Stochastic resource-constrained project scheduling*. PhD thesis, TU Berlin, Berlin, Germany, 2001.

[53] F. Stork and M. Uetz. On the generation of circuits and minimal forbidden sets. *Mathematical Programming*, 102:185–203, 2005.

[54] J.D. Wiest and F.K. Levy. *A Management Guide to PERT/CPM with GERT/PDM/DCPM and other Networks*. Prentice-Hall, 1969.