View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Dynamic Algorithms for Recoverable Robustness Problems*

S. Cicerone[1], G. Di Stefano[1], M. Schachtebeck[2], and A. Schöbel[2]

[1] Department of Electrical and Information Engineering, University of L'Aquila, Italy. {cicerone,gabriele}@ing.univaq.it
[2] Institute for Numerical and Applied Mathematics, Georg-August-University Göttingen, Germany. {schachte,schoebel}@math.uni-goettingen.de

Abstract. Recently, the *recoverable robustness model* has been introduced in the optimization area. This model allows to consider disruptions (input data changes) in a unified way, that is, during both the strategic planning phase and the operational phase. Although the model represents a significant improvement, it has the following drawback: we are typically not facing only one disruption, but many of them might appear one after another. In this case, the solutions provided in the context of the recoverable robustness are not satisfying.
In this paper we extend the concept of recoverable robustness to deal not only with one single recovery step, but with arbitrarily many recovery steps. To this aim, we introduce the notion of *dynamic recoverable robustness problems*. We apply the new model in the context of timetabling and delay management problems. We are interested in finding efficient dynamic robust algorithms for solving the timetabling problem and in evaluating the *price of robustness* of the proposed solutions.

Key words: Robustness, optimization problems, dynamic algorithms, timetabling, delay management.

1 Introduction

In many applications of optimization, the input data is subject to uncertainties and *disruptions* (input data changes). Thus, in most cases, it is desirable not to have a solution that is optimal for the undisturbed input data, but that is feasible even for disturbed input – at the cost of optimality.

Disruptions have to be considered both in the *strategic planning* phase and in the *operational* phase. The latter phase aims to have immediate reaction to disruptions that can occur when the system is running, while the former one aims to plan how to optimize the use of the available resources according to some objective function before the system starts operating.

To face disruptions in the operational phase, the approaches used are mainly based on the concept of *online algorithms* [5]. An online recovery strategy has

* Work partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

to be developed when unpredictable disruptions in daily operations occur, and before the entire sequence of disruptions is known. The goal is to react fast, while retaining as much as possible of the quality of an optimal solution, that is, a solution that would have been achieved if the entire sequence of disruptions was known in advance.

To face disruptions in the strategic planning phase, the approaches used are mainly based on *stochastic programming* and *robust optimization*.

Within stochastic programming (e.g., see [4, 14, 16]), there are two different approaches: *chance constrained programming* aims to find a solution that satisfies the constraints in most scenarios (i.e. with a high probability) instead of satisfying them for all possible realizations of the random variables, while in *multi-stage stochastic programming*, an initial solution is computed in the first step, and each time when some new random data is revealed, a recourse action is taken. However, stochastic programming requires detailed knowledge on the probability distributions of the random variables.

In robust optimization (e.g., see [1–3, 7]), the objective – in contrast to stochastic programming – is purely deterministic. In the concept of *strict robustness*, the solution has to be feasible for all admissible scenarios from a set of input scenarios. The solution gained by this approach can then be fixed since by construction it needs not to be changed when disturbances occur. However, as the solution is fixed independently of the actual scenario, robust optimization leads to solutions that are too conservative and thus too expensive in many applications. One approach to compensate this disadvantage is the idea of *light robustness* introduced in [8]. This approach adds slacks to the constraints. A solution is considered as robust if it satisfies these relaxed constraints.

All the approaches above do not allow to consider disruptions in a unified way, that is, for both the strategic planning and the operational phases. Recently, a first contribution in this direction has been proposed in [15], where the *recoverable robustness* model has been presented. It starts from the practical point of view that a solution is robust if it can be recovered easily in case of a disturbance. This means the solution no longer has to be feasible for all possible scenarios, but a recovery phase is allowed to turn an infeasible solution into a feasible one. However, some limitations on the recovery phase have to be taken into account. For example, the recovery should be quick enough and the quality of the recovered solution should not be too bad.

The initial model of [15] has been extended and applied to shunting problems in [6]. There, the *price of robustness* is defined as the maximum ratio between the cost of the provided robust solution and the optimal solution. According to their price, robust algorithms may also be *exact* or *optimal*.

Although the recoverable robustness model represents a significant improvement in the optimization area, it has the following drawback: We are typically not facing only one disruption, but many of them might appear one after another. In this case, the solutions provided in the context of the recoverable robustness are not satisfying. In this paper we extend the concept of recoverable robustness presented in [6] to deal not only with one single recovery step, but with

arbitrarily many recovery steps. To this aim, we introduce the class DRRP(σ), $\sigma \in \mathbb{N}$, containing all the *dynamic recoverable robustness problems* which have to be solved against σ possible disruptions, appearing one by one. The model in [6] captures exactly the problems in DRRP(1), that is, the *static* recoverable robustness problems.

A concrete example of real world systems, where our model plays an important role, is the *timetable planning*. It arises in the strategic planning phase for transportation systems, and it requires to compute a timetable with e.g. minimal passenger waiting times. However, *many* disturbing events (caused by delays) might occur during the operational phase, and they might completely change the schedule. The problem of deciding which connections from a delayed train to a connecting train should be guaranteed is known in the literature as *delay management problem* [13, 17, 18]. This problem has been shown to be NP-hard in the general case, while it is polynomial solvable in particular cases (see [9–12, 17, 18]).

In this paper, we apply the recoverable robustness model in the context of timetabling and delay management problems. We are interested in finding efficient dynamic robust algorithms for solving the timetabling problem and in evaluating the price of robustness of the proposed solutions. In detail, we take two particular timetabling problems and turn them into problems in DRRP(σ) by defining specific modifications and recovery strategies. For one of such problems we show that finding a solution which minimizes the objective function of the corresponding timetable problem is NP-hard. In general, we propose dynamic robust algorithms and evaluate their prices of robustness. We also prove that such algorithms are optimal with respect to some specific instances.

The remainder of this paper is structured as follows: In Section 2, we show how the concept of recoverable robustness from [15] can be extended to the dynamic framework. Section 3 shows how this framework can be applied to the delay-resistant timetables. In Sections 4 and 5, we propose dynamic robust algorithms, evaluate their prices of robustness and prove optimality in specific instances.

Due to space limitations, some proofs have been omitted.

2 The model

In this section, we extend the model concerning robustness for optimization problems introduced in [6, 15]. We consider minimization problems P characterized by the following parameters:

- I, the set of instances of P;
- $F(i)$, the set of all (potential) feasible solutions for $i \in I$;
- $f\colon S \to \mathbb{R}^{>0}$, the objective function of P that has to be minimized, where $S = \bigcup_{i \in I} F(i)$.

In the dynamic robust optimization problem, we want to find a *robust plan* for some given initial instance $i \in I$ of P. Additional concepts to describe the robustness for the minimization problem P are needed:

- $M : I \to 2^I$ – a *modification* function for instances of P. This function models disturbances of the current scenario due to the following case. If $i \in I$ is the considered input (or scenario) of problem P, a *disturbance* is meant as a modification of i leading to another input scenario $i' \in I$. Such a modification i' depends on the current input i. In order to model this fact, we define the set $M(i)$ as the set of all instances which are modifications of the instance i, i.e. instances that can occur if i is disturbed. Note that the set of modifications may also depend on other information, e.g. on the data of the initial instance.
  Let s be the planned solution for the input i. When a disturbance $i' \in M(i)$ occurs, a new solution $s' \in F(i')$ has to be recomputed for P.
- σ – maximum number of expected modifications. In a practical scenario, several disruptions $i^1, i^2, \ldots, i^\sigma$ may occur. In this case, a task is to devise recovery algorithms that can recompute the solution for P after *each* disruption.
- $\mathbb{A}_{rec}$ – a class of *recovery algorithms* for P. Each element $A_{rec} : S \times I \to S$ works as follows: given a solution $s^0 \in S$ of P (for the current instance i^0) and a modification $i^1 \in M(i^0)$, then $A_{rec}(s^0, i^1) = s^1$, where $s^1 \in F(i^1) \subseteq S$ represents the recovered solution for P. We remark that s^0 and i^1 define the minimal amount of information necessary to recompute the solution. However, for specific cases, A_{rec} could require additional information. In general, when A_{rec} is used at the k-th step, it can use everything that has been processed in the previous steps (in particular, $i^0, ..., i^{k-1}$, and $s^0, ..., s^{k-1}$).

  In general, a class of recovery algorithms $\mathbb{A}_{rec}$ is defined in terms of some kind of *limitation*. In what follows we provide two examples for the class $\mathbb{A}_{rec}$.

  $\mathbb{A}^1_{rec}$**:** this class is based on a constraint on the solutions provided by the recovery algorithm. In particular, the new (recovered) solutions computed by an algorithm must not deviate too much from the original solution s, according to a distance measure d. Formally: given a real number $\Delta \in \mathbb{R}$ and a distance function $d : S \times S \to \mathbb{R}$, we define $\mathbb{A}^1_{rec}$ as the class of algorithms A_{rec} that satisfy the following constraint:
  - $\forall i \in I$, $\forall s \in F(i)$, $\forall i' \in M(i)$, $d(s, A_{rec}(s, i')) \leq \Delta$.

  $\mathbb{A}^2_{rec}$**:** this class is formulated by bounding the computational power of recovery algorithms. Formally: given a function $f : I \times S \times I \to \mathbb{N}$, we define $\mathbb{A}^2_{rec}$ as the class of algorithms A_{rec} that satisfy the following constraint:
  - $\forall i \in I$, $\forall s \in F(i)$, $\forall i' \in M(i)$, $A_{rec}(s, i')$ can be computed in $O(f(i, s, i'))$ time.

2.1 Static model

We first recall the basic definitions concerning robustness for optimization problems introduced in [6, 15].

Definition 1. [6] *A* recoverable robustness problem *is defined by the triple* $(P, M, \mathbb{A}_{rec})$. *All the recoverable robustness problems form the class* RRP.

Definition 2. [6] *Let* $\mathcal{P} = (P, M, \mathbb{A}_{rec})$ *be an element of* RRP. *Given an instance* $i \in I$ *for* P, *an element* $s \in F(i)$ *is a* feasible solution *for* i *with respect to* $\mathcal{P}$ *if and only if the following relationship holds:*

$$\exists A_{rec} \in \mathbb{A}_{rec} : \forall i' \in M(i),\ A_{rec}(s, i') \in F(i').$$

In other words, $s \in F(i)$ is feasible for i with respect to $\mathcal{P}$ if it can be *recovered* by applying some algorithm $A_{rec} \in \mathbb{A}_{rec}$ for each possible disruption $i' \in M(i)$. Hence, s is called *robust* solution. We define $F_{\mathcal{P}}(i)$ as the set of robust solutions for i with respect to $\mathcal{P}$.

Definition 3. [6] *Given* $\mathcal{P} = (P, M, \mathbb{A}_{rec}) \in$ RRP, *a* robust algorithm *for* $\mathcal{P}$ *is any algorithm* $A_{rob} : I \to S$ *such that, for each* $i \in I$, $A_{rob}(i)$ *is robust for* i *with respect to* $\mathcal{P}$.

2.2 Dynamic model

Here we extend the static model in order to deal with a sequence of $\sigma \geq 1$ modifications.

Definition 4. *A* dynamic recoverable robustness problem *is defined by* $(P, M, \mathbb{A}_{rec}, \sigma)$, $\sigma \in \mathbb{N}$. *The class* DRRP(σ) *contains all the problems that have to be solved against* σ *possible disruptions.*

Definition 5. *Let* $\sigma \in \mathbb{N}$ *and* $\mathcal{P} = (P, M, \mathbb{A}_{rec}, \sigma)$ *be an element of* DRRP(σ). *A pair of algorithms* (A_{rob}, A_{rec}) *is called* dynamic robust recovery pair *for* σ *and* $\mathcal{P}$ *if for each instance* $i^0 \in I$, $A_{rec} \in \mathbb{A}_{rec}$ *and the following relationships hold:*

$$s^0 := A_{rob}(i^0) \in F(i^0) \tag{1}$$

$$s^k := A_{rec}(s^{k-1}, i^k) \in F(i^k),\ \forall i^k \in M(i^{k-1}), \forall k \in [1..\sigma], \tag{2}$$

i.e. in the k*-th step, for any possible modification* $i^k \in M(i^{k-1})$ *and for any feasible solution* s^{k-1} *computed in the previous step, the output* s^k *of algorithm* A_{rec} *is a feasible solution for* i^k *with respect to* $\mathcal{P}$.

Note DRRP(1) = RRP. As a consequence, we refer to a *static problem* as a problem in DRRP(1). We use the notation $F_{\mathcal{P}}(i)$ to represent all robust solutions for i with respect to $\mathcal{P} \in$ DRRP(σ) If $\mathbb{A}_{rec}$ is the class of algorithms that never change the solution s for the input i, i.e., if each algorithm $A_{rec} \in \mathbb{A}_{rec}$ satisfies

$$\forall i \in I,\ \forall s \in S,\ \forall i' \in M(i),\ A_{rec}(s, i') = s,$$

then *dynamic recovery robustness* reduces to *strict robustness*. In this case, a robust algorithm A_{rob} for $\mathcal{P}$ must provide a solution s^0 for i^0 such that, for each possible modification $i^k \in M(i^{k-1})$, we have $s^0 \in F_{\mathcal{P}}(i^k)$ for all $k \in [1..\sigma]$. The meaning is the following: If A_{rec} has no recovery capability, then A_{rob} has to find solutions that "absorb" *any* possible sequence of disruptions.

2.3 Price of robustness

For every instance $i \in I$, the price of robustness of A_{rob} is given by the maximum ratio between the cost of the solution provided by A_{rob} and the optimal solution.

Definition 6. *The* price of robustness *of a robust algorithm A_{rob} for a problem $\mathcal{P} \in \mathrm{DRRP}(\sigma)$ is*

$$P_{rob}(\mathcal{P}, A_{rob}) = \max_{i \in I} \left\{ \frac{f(A_{rob}(i))}{\min\{f(x) : x \in F(i)\}} \right\}.$$

The price of robustness of a problem $\mathcal{P} \in \mathrm{DRRP}(\sigma)$ is given by the minimum price of robustness among all possible robust algorithms. Formally,

Definition 7. *The* price of robustness *of a problem $\mathcal{P} \in \mathrm{DRRP}(\sigma)$ is given by*

$$P_{rob}(\mathcal{P}) = \min\{P_{rob}(\mathcal{P}, A_{rob}) : A_{rob} \text{ is a robust algorithm for } \mathcal{P}\}.$$

If there are many robust algorithms possible for $\mathcal{P}$, we want to identify the "best" one:

Definition 8. *Let $\mathcal{P} \in \mathrm{DRRP}(\sigma)$ and let A_{rob} be a robust algorithm for $\mathcal{P}$. Then,*

- *A_{rob} is* exact *if $P_{rob}(\mathcal{P}, A_{rob}) = 1$;*
- *A_{rob} is* $\mathcal{P}$-optimal *if $P_{rob}(\mathcal{P}, A_{rob}) = P_{rob}(\mathcal{P})$. The solution computed by a $\mathcal{P}$-optimal algorithm is called $\mathcal{P}$-optimal.*

Note that the definition of an exact robust algorithm refers to the problem $\mathcal{P}$. We state a simple observation concerning the price of robustness.

Lemma 1. *For fixed P, M, and $\mathbb{A}_{rec}$, consider a family of problems $\mathcal{P}_\sigma = (P, M, \mathbb{A}_{rec})$ for different values of σ, i.e. these problems vary in the expected number of recoveries only. For $\sigma_1 < \sigma_2$, we have*

- *$F_{\mathcal{P}_{\sigma_2}}(i) \subseteq F_{\mathcal{P}_{\sigma_1}}(i)$ for all instances $i \in I$,*
- *$P_{rob}(\mathcal{P}_{\sigma_1}) \leq P_{rob}(\mathcal{P}_{\sigma_2})$, i.e. the price of robustness grows in the number of expected recoveries.*

Proof. Let $s \in F_{\mathcal{P}_{\sigma_2}}$, i.e. there exist (A_{rob}, A_{rec}) such that (2) holds for all $k = 1, \ldots, \sigma_2$, hence also for all $k = 1, \ldots, \sigma_1$. This yields $s \in F_{\mathcal{P}_{\sigma_1}}$. Moreover, it shows that A_{rob} is robust for $\mathcal{P}_{\sigma_1}$ if A_{rob} is robust for $\mathcal{P}_{\sigma_2}$, hence also the second statement holds. □

3 Application to Delay Management

In this section we apply the concept of dynamic robustness to a (simplified variant) of the *timetabling* problem in public transportation. Timetabling is a real-world problem which suffers a lot from disturbances or delays. Whenever an unexpected source delay occurs, the timetable has to be recovered to a *disposition timetable*. Recovering a given timetable in case of delays is known as the *delay management problem*. Usually, there is not only one (source) delay but many such delays during a day which occur one after another. Our concept of *dynamic recoverable robustness* is hence important for this application.

3.1 Notation and Definition

We first introduce the specific timetabling problem which we will consider, and describe the delay scenarios and the restrictions for the recovery we are looking at. We then investigate the recovery restrictions in detail.

The Initial Planning Problem. Let $G = (V, A)$ be a directed acyclic graph (DAG) with one specified node v_1 such that there exists a directed path from v_1 to each other node.

Referring to the notation common in timetabling in public transportation, let us call the nodes in V *events* and the edges A *activities*. The nodes refer to arrival and departure events and the activities to driving, waiting and changing activities. Moreover, we say a DAG $G = (V, A)$ is a *tree*, if it is an out-tree to some source node v_1 (i.e. the path from v_1 to each other node is unique), and a *linear graph* if $V = \{v_1, v_2, \ldots, v_n\}$ and $A = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$.

In order to define the timetabling problem, let us assume that we have given weights $w_u \in \mathbb{R}$ for each $u \in V$ representing the importance of the corresponding event and lower bounds $L_a^0 \in \mathbb{R}^{>0}$ indicating the minimal duration that is needed for activity $a \in \mathcal{A}$.

We are looking for a timetable $\pi : V \to \mathbb{R}^{\geq 0}$ assigning a point of time to each event $u \in V$. π is feasible if it respects the minimal duration of each activity (see (3)), i.e. our initial planning problem P can be stated as

$$(P) \quad \min f(\pi) = \sum_{u \in V} w_u \pi_u \qquad \text{s.t.}$$

$$\pi_v - \pi_u \geq L_a^0 \quad \forall a = (u, v) \in A \tag{3}$$

$$\pi_u \in \mathbb{R}^{\geq 0} \quad \forall u \in V. \tag{4}$$

An instance i of P is specified by $i = (G, w, L^0)$. Given $a = (u, v) \in A$, the amount $\pi_v - \pi_u - L_a$ is called *slack time* for the activity a. (P) can be solved in polynomial time by linear programming. We will consider two special cases of (P), both having the same constraints (3) and (4), but differ in their objective functions.

P1 Here we consider the objective which is usually used in timetabling, namely to minimize the sum of all activity lengths (or equivalently to minimize the slack times). The objective of this problem is

$$(P1) \quad \min f(\pi) = \sum_{a=(u,v) \in \mathcal{A}} w_a(\pi_v - \pi_u),$$

with $w_a \in \mathbb{R}^{\geq 0}$ for all $a \in \mathcal{A}$. It is a special case of (P), namely if $w_u = \sum_{a=(v,u) \in \mathcal{A}} w_a - \sum_{a=(u,v) \in \mathcal{A}} w_a$ for each $u \in V$.

P2 Here we consider the problem (P) but require $w_u \geq 0$ for all $u \in V$.

Note that (P2) can be efficiently solved by the forward phase of the critical path method of project planning (CPM): given an instance $i = (G, w, L^0)$ of (P2), the solution $\pi = CPM(i)$ for i can be computed as follows:

$$CPM(i) = \begin{cases} \pi_v = 0 & \text{if } v = v_1 \\ \pi_v = \max\{\pi_u + L_a \ : \ a = (u,v) \in A\} & \text{otherwise} \end{cases}$$

The following lemmata can be proven by induction.

Lemma 2. *Given an instance $i = (G, w, L^0)$ of (P1), $CPM(i)$ is optimal if G is a tree.*

Note that this needs not to be true if G is not a tree: It often makes sense to schedule events that do not belong to the critical path later than necessary to avoid slack on activities with high weights w_a.

Lemma 3. *Given an instance $i = (G, w, L^0)$ of (P2), $CPM(i)$ is optimal.*

In the next sections we will use these lemmata to give a general solution approach for some robustness versions of both (P1) and (P2).

The Static Problem. We first describe the problem $\mathcal{P}_{\sigma=1} = (P, M, \mathbb{A}_{rec}, 1) \in$ DRRP(1), where P corresponds to one of the timetabling problems (P1) or (P2) defined above, while M and $\mathbb{A}_{rec}$ are defined as follows:

- The modification function M for an instance $i^0 = (G, w, L^0)$ and a constant $\alpha \in \mathbb{R}^{>0}$ is defined as:

$$M(i^0) = \left\{(G, w, L^1) : \exists \overline{a} \in \mathcal{A} : L^0_{\overline{a}} < L^1_{\overline{a}} \leq L^0_{\overline{a}} + \alpha \text{ and } L^1_a = L^0_a \forall a \neq \overline{a}\right\}.$$

Using this function, we represent the delay of an activity a by increasing the initial value L^0_a to some value $L^1_a > L^0_a$. The definition ensures that only one delay is allowed (and bounded by α) and that nothing changes on all other activities.

- The feasible set of recovery algorithms $\mathbb{A}_{rec}$. We are going to investigate the following two limitations:

– **limited-events**: here we assume that there are resources to change the time for a limited number of events only. In particular, if π is a solution for P and x^1 is a disposition timetable computed by any recovery algorithm in $\mathbb{A}_{rec}$, then x^1 must satisfy

$$d(x^1, \pi) := \left|\{u \in V : x^1_u \neq \pi_u\}\right| \leq \Delta \tag{5}$$

for some given $\Delta \in \mathbb{N}$.

– **limited-delay**: as second limitation of the recovery algorithm, we again require that x^1 must not deviate "too much" from the initial timetable π, but this time we consider the sum of all deviations of all events. I.e. x^1 must satisfy

$$d(x^1, \pi) := \left\|x^1 - \pi\right\|_1 \leq \Delta \tag{6}$$

for some given $\Delta \in \mathbb{N}$.

In both cases, $\Delta = 0$ means strict robustness.

We are looking for a feasible pair (A_{rob}, A_{rec}) for $\mathcal{P}_{\sigma=1}$ (according to Definition 5). The result of A_{rob} is a robust solution $\pi = A_{rob}(i^0) \in F_{\mathcal{P}_{\sigma=1}}$ for each instance $i^0 = (G, w, L^0)$ of P. It has to satisfy the following constraints:

$$\pi_v - \pi_u \geq L^0_a, \quad \forall a = (u,v) \in A$$
$$\forall\ (G, w, L^1) \in M(i^0),\ \forall u \in V,\ \exists\ x^1_u \in \mathbb{R}^{\geq 0}: \tag{7}$$
$$x^1_v - x^1_u \geq L^1_a, \quad \forall a = (u,v) \in A \tag{8}$$
$$x^1_u \geq \pi_u, \quad \forall u \in V \tag{9}$$
$$d(x^1, \pi) \leq \Delta. \tag{10}$$

In Section 4.1 we show that it is NP-hard to compute the $\mathcal{P}_{\sigma=1}$-optimal solution for (P2) and the limited-delay case.

Concerning A_{rec}, let us assume that π is a solution computed by any robust algorithm A_{rob} with respect to the instance i^0. Let $i^1 = (G, w, L^1) \in M(i^0)$. We know that (7)–(10) hold, i.e. a disposition timetable x^1 exists. It can be computed by an *updating version* of CPM:

$$CPM(i^0, \pi, i^1) = \begin{cases} x^1_v = 0 & \text{if } v = v_1 \\ x^1_v = \max\left\{\pi_v, \max\left\{x^1_u + L^1_a\ :\ a = (u,v) \in A\right\}\right\} & \text{otherwise.} \end{cases}$$

This recovery algorithm computes the disposition timetable x^1 with the minimum value of $d(x^1, \pi)$ for both limitations, i.e. it minimizes $\left|\{u \in V : x^1_u \neq \pi_u\}\right|$ and $\left\|x^1 - \pi\right\|_1$ at the same time. Hence it is able to recover (if a recovery solution exists) or to find out that such a solution does not exist. Additionally, among all timetables satisfying constraints (8)–(10), the recovery algorithm provides the disposition timetable with the optimal value for (P2) and, if G is a tree, also for (P1).

The Dynamic Problem. We already remarked that in real-world operation, we have to expect more than one delay. We hence consider $\mathcal{P}_{\sigma\geq 1} = (P, M, \mathbb{A}_{rec}, \sigma) \in \mathrm{DRRP}(\sigma)$, $\sigma \in \mathbb{N}$. We formalize M and $\mathbb{A}_{rec}$ as follows:

- The modification function for an instance $i^{k-1} = (G, w, L^{k-1})$ and a constant $\alpha \in \mathbb{R}^{>0}$ is:

$$M(i^{k-1}) = \left\{(G, w, L^k) : \exists \overline{a} \in \mathcal{A} : L^k_a = L^{k-1}_a\ \forall a \neq \overline{a} \text{ and } L^0_{\overline{a}} \leq L^k_{\overline{a}} \leq L^0_{\overline{a}} + \alpha\right\}.$$

- $\mathbb{A}_{rec}$ is based on the same two limitations as in the static case. In particular, we require that a solution x^k computed by an algorithm in $\mathbb{A}_{rec}$ satisfies $d(x^k, \pi) \leq \Delta$, where d is defined according to (5) or according to (6).

Let $i^0 = (G, w, L^0)$ be an instance of $\mathcal{P}_{\sigma\geq 1}$. Again, each robust solution satisfies the (generalized) constraints (7)-(10). Analogously to the static case, the updating version of CPM can be used as recovery algorithm.

In Section 4 we address the problem of designing robust algorithms for $\mathcal{P}_{\sigma\geq 1} = (P, M, \mathbb{A}_{rec}, \sigma)$ where $P \equiv P2$ and $\mathbb{A}_{rec}$ is based on limited events

(see Eq. (5)). Similarly, in Section 5 we investigate the case in which $P \equiv P1$ and $\mathbb{A}_{rec}$ is based on the overall delay according to Eq. (6). In both sections we provide robust algorithms by using the following general approach: first, add an additional slack time s_a to the lower bounds L_a^0 of each activity $a \in A$; then, compute an optimal solution of the resulting instance and take it as a robust solution. Formally, we obtain an algorithm Alg_s^+ for each value of s:

ALGORITHM Alg_s^+	
INPUT:	An instance $i^0 = (G, w, L^0)$ of (P)
ALGORITHM:	1. Define $\bar{L}_a := L_a^0 + s$
	2. Solve $\bar{i} = (G, w, \bar{L})$ optimally.

The variant Alg_s^* differs from Alg_s^+ at Step 1: $\bar{L}_a := s \cdot L_a^0$, that is, instead of adding the slack time, all lower bounds are multiplied by some value.

According to Lemma 2, in the case of (P1), Step 2 can be done efficiently by the critical path method CPM when G is a tree. Otherwise, linear programming can be used. According to Lemma 3, in the case of (P2), Step 2 can be done efficiently by the critical path method CPM when G is a DAG.

For the recoverable robustness problems addressed in Sections 4 and 5, algorithms Alg_s^+ and Alg_s^* are robust if s is large enough. Moreover, the price of robustness increases in s. In particular:

- Alg_s^+ is strictly robust if $s \geq \alpha$. Alg_s^* is strictly robust if $s \geq \frac{L_a^0 + \alpha}{L_a^0}$ for all $a \in A$.
- Let Alg_s^+ (Alg_s^*) be robust. Then $\mathsf{Alg}_{s'}^+$ ($\mathsf{Alg}_{s'}^*$) is robust for all $s' \geq s$.
- Let $\mathsf{Alg}_{s_1}^+$ and $\mathsf{Alg}_{s_2}^+$ ($\mathsf{Alg}_{s_1}^*$ and $\mathsf{Alg}_{s_2}^*$), $s_2 \geq s_1$, be robust. Then

$$P_{rob}(\mathcal{P}_{\sigma \geq 1}, \mathsf{Alg}_{s_1}^+) \leq P_{rob}(\mathcal{P}_{\sigma \geq 1}, \mathsf{Alg}_{s_2}^+), \text{ and}$$

$$P_{rob}(\mathcal{P}_{\sigma \geq 1}, \mathsf{Alg}_{s_1}^*) \leq P_{rob}(\mathcal{P}_{\sigma \geq 1}, \mathsf{Alg}_{s_2}^*).$$

According to the above algorithmic approach, in order to minimize the price of robustness, the goal is to find the smallest value for s such that the respective algorithms are robust. However, in Section 4 we also provide robust algorithms based on a different approach (adding slack times only to specific activities).

4 Dynamic recovery with number of events as limitation

In this section we consider the first case of limitation: we have to find a timetable π such that in each recovered solution x^k, the times of up to Δ nodes may deviate from the original timetable, i.e. x^k must satisfy (5) for some given $\Delta \in \mathbb{N}$ and for all $k = 1, 2, \ldots, \sigma$. Moreover, we consider the problem $\mathcal{P}_{\sigma \geq 1}$ based on the initial planning problem (P2).

It is clear that each timetable is robust if $\Delta \geq |V| - 1$, so in the following, we will always assume $\Delta \leq |V| - 2$. If $\sigma > \Delta$, we need strict robustness to get a robust solution:

Lemma 4. *If $\sigma > \Delta$, then a timetable is robust if and only if the slack s satisfies $s_a \geq \alpha$ for each $a \in A$. In this case, we have strict robustness.*

From Lemma 1 we know that $F_{\mathcal{P}_{\sigma\geq 1}}(i^0) \subseteq F_{\mathcal{P}_{\sigma=1}}(i^0) \subseteq F(i^0)$. Since the set of robust solutions in the dynamic case is smaller than the same set in the static case, the price of robustness for $\mathcal{P}_{\sigma\geq 1}$ is smaller than or equal to the price of robustness for $\mathcal{P}_{\sigma=1}$. Now, we present an example showing that even $F_{\mathcal{P}_{\sigma\geq 1}}(i^0) \subsetneqq F_{\mathcal{P}_{\sigma=1}}(i^0) \subsetneqq F(i^0)$ holds.

Example 1. Consider a simple instance $i^0 = (G, w, L^0)$ of (P2), where: $G = (V, A)$ is a linear graph with four events and three activities, $w_u = 1$ for each $u \in V$, and $L^0_a = 1$ for each $a \in A$. Concerning $\mathcal{P}_{\sigma\geq 1}$, we fix $\alpha = 1$ and $\Delta = 1$.

Figure 1 shows different solutions (timetables) for the instance i^0. It is easy to see that timetable π_{CPM} (computed by CPM) is feasible for (P2). Conversely, any delay on the first activity implies that all the subsequent three events must be delayed. Hence, since $\Delta = 1$, then π_{CPM} is not in $F_{\mathcal{P}_{\sigma=1}}(i^0)$.

A solution belonging to $F_{\mathcal{P}_{\sigma=1}}(i^0)$ is π. In fact, each possible delay on the three activities, in order, is recovered by the three disposition timetables x, x', and x'', respectively. Note that these timetables differ from π by at most $\Delta = 1$ events. On the other hand, π is not in $F_{\mathcal{P}_{\sigma\geq 1}}(i^0)$. This fact can be observed by assuming that two delays, both of α time, occur on the first two activities. The best disposition timetable that recovers these delays starting from π is x''', but $d(\pi, x''') = 3 > \Delta$.

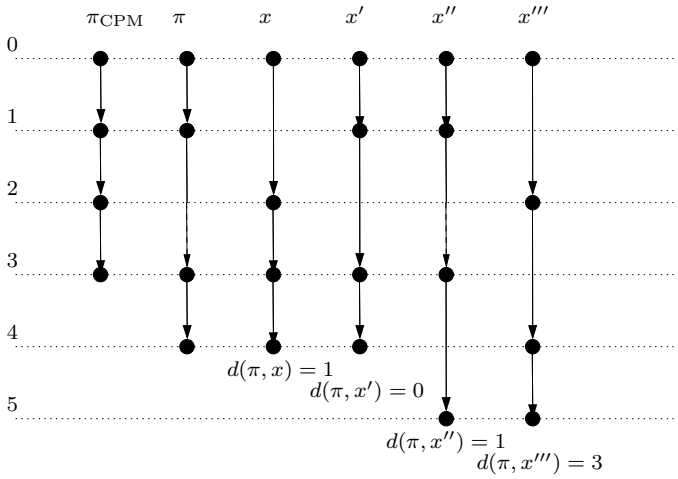

Fig. 1. A graphical representation of different timetables described in Example 1. Bullets represent events and arrows represent activities. Time assigned to each event corresponds to the integer associated to the horizontal line on which the event lies to. The dotted part of arrows represents slack times.

The following lemma implies that $F_{\mathcal{P}_{\sigma\geq 1}}(i^0) = F_{\mathcal{P}_{\sigma=1}}(i^0)$ for $\Delta = 0$.

Lemma 5. *Let $\mathcal{P}_{\sigma=1}$ and $\mathcal{P}_{\sigma\geq 1}$ defined with $\Delta = 0$. If A_{rob} is a robust algorithm for $\mathcal{P}_{\sigma=1}$, then A_{rob} is robust for $\mathcal{P}_{\sigma\geq 1}$.*

Proof. Let A_{rob} be a robust algorithm for $\mathcal{P}_{\sigma=1}$, and let $\pi = A_{rob}(i^0)$. We first show that π assigns a slack time of at least α time to each activity. By contradiction, let us assume that there exists an activity $a = (u, v) \in A$ such that $\pi_v - \pi_u - L_a < \alpha$. Now, if a modification $i^1 \in M(i^0)$ is such that $L^1_a = L_a + \alpha$, namely a delay of α time occurs on a, then $\pi_v - \pi_u < L^1_a$. This means that π is not feasible for i^1, a contradiction for $\Delta = 0$.

Since π assigns a slack of at least α time to each activity, it follows that A_{rob} is a robust algorithm for $\mathcal{P}_{\sigma\geq 1}$. □

4.1 The complexity of computing the price of robustness

We show that the problem of computing the $F_{\mathcal{P}_{\sigma=1}}$-optimal solution is NP-hard for (P2) (and hence also for (P)). This implies that computing the $F_{\mathcal{P}_{\sigma\geq 1}}$-optimal solution is NP-hard.

To capture the concept of *events affected by a delay*, that is, those events that must be postponed as a consequence of a given delay, we give the following definition.

Definition 9. *Given a DAG $G = (V, A)$, a function $s : A \to \mathbb{R}^{\geq 0}$, and a number $\alpha \in \mathbb{R}^{\geq 0}$, a vertex y is* α-influenced *by $(u, v) \in A$ (equivalently (u, v) α-influences y) if there exists a path $p = (u \equiv u_0, v \equiv u_1, \ldots, u_k \equiv y)$ in G such that $\sum_{i=1}^{k} s_{(u_{i-1},u_i)} < \alpha$.*

Remark 1. If x is α-influenced by a according to a path p, then all the vertices belonging to p but the source are α-influenced by a.

In the above definition, function s represents slack times associated to activities able to (partially) absorb a delay. Note that every robust algorithm A_{rob} must provide a timetable π such that each arc cannot influence more than Δ vertices, otherwise there exists no A_{rec} algorithm. In order to show the NP-hardness of computing the $F_{\mathcal{P}_{\sigma=1}}$-optimal solution, we introduce a corresponding decision problem called MRS (Minimum Robust Solution) that uses the concept of α-influence.

MRS Problem	
GIVEN:	DAG $G = (A, V)$, a function $L : A \to \mathbb{R}^{>0}$, a function $w : V \to \mathbb{R}^{\geq 0}$, and three numbers $\alpha \in \mathbb{R}^{>0}$, $\Delta \in \mathbb{N}, K \in \mathbb{N}$
PROBLEM:	Find a timetable $\pi : V \to \mathbb{R}^{\geq 0}$ such that each arc α-influences at most Δ vertices, according to the function $s : A \to \mathbb{R}$ defined as $s_{a=(i,j)} = \pi_j - \pi_i - L_a$, and such that $\sum_{u\in V} w_u \pi_u \leq K$.

Theorem 1. MRS *is NP-complete for any fixed $\Delta \geq 3$.*

Proof. Omitted. □

Corollary 1. *The problem of computing $P_{rob}(\mathcal{P}_{\sigma\geq 1})$ with number of events as limitation is NP-hard.*

4.2 Robust algorithms for $\sigma = 1$ on an arbitrary DAG.

We use the idea described in Section 3.1 (page 10) and add a slack to the minimal durations of all activities or multiply them with some number >1. To this end, let $i = (G, L, w)$, $\alpha \in \mathbb{R}^{\geq 0}$ and $\gamma \in \mathbb{R}^{>0}$, we denote $i_\alpha = (G, L + \alpha, w)$ and $i_\gamma = (G, \gamma L, w)$. We use the critical path method to define robust solutions for $\mathcal{P}_{\sigma=1}$ and $\mathcal{P}_{\sigma\geq 1}$. In particular, we use the algorithms Alg^+_α and Alg^*_γ defined as

- $\mathsf{Alg}^+_\alpha(i) = \mathit{CPM}(i_\alpha)$;
- $\mathsf{Alg}^*_\gamma(i) = \mathit{CPM}(i_\gamma)$.

Let L_{min} be the minimum value assigned by the function L with respect to all the possible instances of $\mathcal{P}_{\sigma\geq 1}$. In the following, let α be as in the definition of the modification function M of $\mathcal{P}_{\sigma\geq 1}$ and $\gamma = (1 + \frac{\alpha}{L_{min}})$. We use α and γ to get concrete instances of Alg^+_α and Alg^*_γ. According to the proof of Lemma 5, if $\mathcal{P}_{\sigma=1}$ is defined with $\Delta = 0$, then every robust algorithm for $\mathcal{P}_{\sigma=1}$ must provide solutions that assign a slack time of at least α to each activity. Then, it follows that Alg^+_α is a robust algorithm for $\mathcal{P}_{\sigma=1}$. To show that also Alg^*_γ is a robust algorithm for $\mathcal{P}_{\sigma=1}$, it is sufficient to observe that for each activity $a \in A$,

$$\gamma L_a = (1 + \frac{\alpha}{L_{min}})L_a = L_a + \alpha\frac{L_a}{L_{min}} \geq L_a + \alpha.$$

The following lemma shows the price of robustness of Alg^*_γ.

Lemma 6. *Let $\mathcal{P}_{\sigma=1}$ be defined with $\Delta = 0$. Then, $P_{rob}(\mathcal{P}_{\sigma=1}, \mathsf{Alg}^*_\gamma) = 1 + \alpha/L_{min}$.*

Proof. By definition,

$$P_{rob}(\mathcal{P}_{\sigma=1}, \mathsf{Alg}^*_\gamma) = \max_{i\in I}\left\{\frac{f(\mathsf{Alg}^*_\gamma(i))}{\min\{f(x) : x \in F(i)\}}\right\}.$$

Let $i = (G, L, w)$ be an instance of $\mathcal{P}_{\sigma=1}$. Denoting by π^γ the solution provided by $\mathsf{Alg}^*_\gamma(i)$, and by π the solution provided by $\mathit{CPM}(i)$, then

$$P_{rob}(\mathcal{P}_{\sigma=1}, \mathsf{Alg}^*_\gamma) = \max_{i=(G,L,w)\in I}\frac{\sum_{u\in V} w_u\pi^\gamma_u}{\sum_{u\in V} w_u\pi_u}.$$

Now we show that for each $v \in V$, $\pi^\gamma_v = \gamma\pi_v$. By contradiction, let $v \in V$ be an event such that $\pi^\gamma_v \neq \gamma\pi_v$ and π^γ_v is minimum. Clearly, v must be different from v_1 and hence there exists an activity $a = (u, v) \in A$ such that $\pi^\gamma_v = \pi^\gamma_u + \gamma L_a$. As

π_v^γ is minimum and $\pi_u^\gamma < \pi_v^\gamma$, then $\pi_u^\gamma = \gamma\pi_u$. It follows $\pi_v^\gamma = \gamma\pi_u + \gamma L_a = \gamma\pi_v$, a contradiction. Hence,

$$P_{rob}(\mathcal{P}_{\sigma=1}, \mathsf{Alg}_\gamma^*) = \max_{i=(G,L,w)\in I} \frac{\sum_{u\in V} w_u \pi_u^\gamma}{\sum_{u\in V} w_u \pi_u} = \max_{i=(G,L,w)\in I} \frac{\sum_{u\in V} w_u \gamma\pi_u}{\sum_{u\in V} w_u \pi_u} = \gamma.$$

□

Lemma 7. *For each instance $i \in I$, $f(\mathsf{Alg}_\alpha^+(i)) \leq f(\mathsf{Alg}_\gamma^*(i))$.*

Proof. Let $i = (G, L, w) \in I$. Let us denote by π^γ the solution provided by $\mathsf{Alg}_\gamma^*(i)$, by π^α the solution provided by $\mathsf{Alg}_\alpha^+(i)$, and by π the solution provided by $CPM(i)$. To prove the statement, it is sufficient to show that

$$\pi_u^\alpha \leq \pi_u^\gamma, \ \forall\ u \in V.$$

By contradiction, let us assume that there exists an event u such that $\pi_u^\alpha > \pi_u^\gamma$. In the proof of Lemma 6, it is shown that $\pi_u^\gamma = \gamma\pi_u$. Then,

$$\pi_u^\alpha > \pi_u^\gamma = \gamma\pi_u = (1 + \frac{\alpha}{L_{min}})\pi_u.$$

Since, by contradiction hypothesis $\pi_u^\alpha > \pi_u^\gamma$, then $u \neq v_1$. It follows that there exists a path $(v_1, \ldots, u)$ in G such that its length ℓ is greater than 0. By definition of Alg_α^+, then

$$\pi_u^\alpha = \pi_u + \alpha\ell.$$

In conclusion,

$$\pi_u^\alpha = \pi_u + \alpha\ell > (1 + \frac{\alpha}{L_{min}})\pi_u = \pi_u + \alpha\frac{\pi_u}{L_{min}}.$$

It follows that $\ell L_{min} > \pi_u$, a contradiction. □

Lemmata 6 and 7 imply the following results.

Corollary 2. *Let $\mathcal{P}_{\sigma=1}$ be defined with $\Delta = 0$. Then, $P_{rob}(\mathcal{P}_{\sigma=1}, \mathsf{Alg}_\alpha^+) \leq 1 + \alpha/L_{min}$.*

4.3 Robust algorithms for $\sigma = 1$ on a linear graph.

In this section, we present an algorithm that computes an optimal robust timetable for the problem (P2) for the case $\sigma = 1$ on a linear graph. The idea of the algorithm is to add each slack "as late as possible". Let $V = \{v_1, \ldots, v_{|V|}\}$ be ordered such that $A = \{a_1 = (v_1, v_2), \ldots, a_{|A|} = (v_{|V|-1}, v_{|V|})\}$. Define s^α by

$$s_{a_j}^\alpha := \begin{cases} \alpha & \text{if } (\Delta+1)|j \\ 0 & \text{else} \end{cases} \tag{11}$$

for all arcs $a_j \in A$. We then add s_a^α to L_a^0 for each $a \in A$ and calculate a solution of (P2) by applying CPM. We denote this algorithm by $\mathsf{Alg}_{s^\alpha}^+$. The following lemma states that a timetable π is robust if and only if the slack time of each $\Delta + 1$ consecutive arcs is large enough to let vanish the delay.

Lemma 8. *If $\sigma = 1$ and $\Delta \leq |V| - 2$, a timetable π for a linear graph G is robust if and only if*

$$\sum_{k=0}^{\Delta} s_{a_{j+k}} \geq \alpha \quad \text{for each } j = 1, \ldots, |A| - \Delta. \tag{12}$$

Theorem 2. *$\mathsf{Alg}^+_{s^\alpha}$ is an optimal robust algorithm for $\mathcal{P}_{\sigma=1}$ based on the initial problem (P2).*

Proof. Omitted. □

Corollary 3. *If G is a linear graph, there exists a linear time algorithm that computes $\mathcal{P}_{\sigma=1}$-optimal solutions.*

Proof. Running algorithm $\mathsf{Alg}^+_{s^\alpha}$ on a linear graph needs time $\mathcal{O}(|A|)$, checking whether the output satisfies $\sum_{u \in V} w_u \pi_u \leq K$ needs time $\mathcal{O}(|V|)$. As $\mathsf{Alg}^+_{s^\alpha}$ is an optimal robust algorithm, it finds a feasible timetable if and only if (MRS) is feasible. □

4.4 Robust algorithms for arbitrary σ on a linear graph.

We now present an algorithm for an arbitrary σ if G is a linear graph. It assigns the same slack

$$s^* = \min \left\{ \alpha, \frac{\sigma\alpha}{\Delta + 1} \right\} \tag{13}$$

to each arc. In the following, we will show that $\mathsf{Alg}^+_{s^*}$ is robust and that it is optimal compared to all robust algorithms that add an equal slack s to all arcs. We need the following two lemmata for the proof:

Lemma 9. *If $s_a < \alpha$ for all arcs $a \in A$, then the number of nodes affected by a single delay of $\sigma\alpha$ on arc $a_j = (v_j, v_{j+1})$ is equal to the number of nodes affected by σ single delays of α on the σ consecutive arcs $a_{j+k} = (v_{j+k}, v_{j+k+1})$, $k = 0, \ldots, \sigma - 1$.*

Proof. Let $s_a < \alpha$ for all arcs $a \in A$. If, on the one hand, a_j is delayed by $\sigma\alpha$, then v_{j+1} has a delay of $\sigma\alpha - s_{a_j}$, v_{j+2} has a delay of $\sigma\alpha - s_{a_j} - s_{a_{j+1}}$ and so on, and $v_{j+\sigma}$ has a delay of $\sigma\alpha - \sum_{k=0}^{\sigma-1} s_{a_{j+k}}$. If, on the other hand, $a_j, \ldots, a_{j+\sigma-1}$ are delayed by α, then v_{j+1} has a delay of $\alpha - s_{a_j}$, v_{j+2} has a delay of $2\alpha - s_{a_j} - s_{a_{j+1}}$ and so on, and $v_{j+\sigma}$ has a delay of $\sigma\alpha - \sum_{k=0}^{\sigma-1} s_{a_{j+k}}$. As $s_a < \alpha$ for all arcs $a \in A$, all these delays are positive.

So in both cases, the nodes $v_{j+1}, \ldots, v_{j+\sigma}$ are affected, and as the delay of $v_{j+\sigma}$ is the same in both cases, the total number of subsequent affected nodes is the same, too. □

Lemma 10. *If all arcs $a \in A$ have the same slack $s_a = s$, then the number of nodes affected by σ delays of α on σ consecutive arcs is always greater than or equal to the number of nodes affected by σ delays of α on σ non-consecutive arcs.*

Proof. Omitted. □

Now, we can prove that $\mathsf{Alg}_{s^*}^+$ is robust and that it is optimal for (P2):

Theorem 3. *Let G be a linear graph. Assume that we add the same slack time s to all arcs. Then Alg_s^+ is a robust algorithm for $\mathcal{P}_{\sigma\geq 1}$ if and only if $s \geq s^*$. If $s > s^*$ and $|V| > 1$, then $f(\mathsf{Alg}_s^+(i)) > f(\mathsf{Alg}_{s^*}^+(i))$ if $w_u \geq 0$ for all nodes $u \in V$ and $w_u > 0$ for at least one node $u \in V$.*

Proof. Omitted. □

Theorem 4. *Let s^* be defined as in Eq. (13), and let G be a linear graph. If, for each $(G, w, L^0) \in I$, $w_a > 0$ for at least one $a \in A$, then $P_{rob}(\mathcal{P}_{\sigma\geq 1}, \mathsf{Alg}_{s^*}^+) \leq 1 + s^*/L_{min}$.*

Proof. Omitted. □

5 Dynamic recovery with sum of delays as limitation

In this section we consider the second case of limitation: We have to find a timetable π such that each recovered solution x^k must not deviate too much from the initial timetable π, i.e. x^k must satisfy

$$d(x^k, \pi) := \left\|x^k - \pi\right\|_1 \leq \Delta \tag{14}$$

for some given $\Delta \in \mathbb{N}$ and for all $k = 1, 2, \ldots, \sigma$. Throughout this section, we consider the problem $\mathcal{P}_{\sigma\geq 1}$ based on the initial planning problem (P1) which implies that all weights w_a are nonnegative.

Again, our strategy to make a timetable robust against delays is to add an amount s of slack time to all the arcs, i.e. to use the algorithm Alg_s^+.

We first investigate how much we loose in the optimal solution if we use Alg_s^+ instead of an algorithm that computes the optimal (but not robust) solution of (P1), i.e. without the additional slack s. Again, let L_{min} be the minimum value assigned by the function L with respect to all the possible instances of $\mathcal{P}_{\sigma\geq 1}$.

Lemma 11. *Let G be a tree and let $w_a > 0$ for at least one $a \in A$. If Alg_s^+ is robust, its price of robustness is $P_{rob}(\mathcal{P}_{\sigma\geq 1}, \mathsf{Alg}_s^+) \leq 1 + s/L_{min}$.*

Proof.

$$\begin{aligned}
P_{rob}(\mathcal{P}_{\sigma\geq 1}, \mathsf{Alg}_s^+) &= \max_{i=(G,w,L^0)\in I} \frac{\sum_{a=(u,v)\in A} w_a(L_a^0 + s)}{\sum_{a=(u,v)\in A} w_a L_a^0} \\
&= 1 + s \cdot \max_{i=(G,w,L^0)\in I} \frac{\sum_{a=(u,v)\in A} w_a}{\sum_{a=(u,v)\in A} w_a L_a^0} \\
&\leq 1 + s \cdot \max_{i=(G,w,L^0)\in I} \frac{\sum_{a=(u,v)\in A} w_a}{\sum_{a=(u,v)\in A} w_a L_{min}} \\
&\leq 1 + s/L_{min}.
\end{aligned}$$

□

Now we discuss how much slack time s is needed to guarantee robustness of Alg_s^+. Our first result deals with strict robustness, i.e. if $\Delta = 0$. In this case we have to make sure that any delay can be compensated by the slack time on the corresponding edge. Since L_a^k never differs from L_a^0 by more than α in any scenario, it suffices to add an additional slack of α to each L_a^0 for all $a \in A$. Then the resulting disposition timetable in each step equals the original timetable π, i.e. a recovery step is in fact not necessary.

Lemma 12. *The algorithm Alg_α^+ is strictly robust (i.e. it is robust for the case $\Delta = 0$) for any graph G. Furthermore, if G is a tree, its price of robustness is $P_{rob}(\mathcal{P}_{\sigma\geq 1}, \mathsf{Alg}_\alpha^+) \leq 1 + \alpha/L_{min}$.*

Proof. The robustness of Alg_α^+ is clear. The price of robustness follows from Lemma 11. □

Now we turn our attention to the case $\Delta > 0$, but first only look at one recovery step (i.e. $\sigma = 1$). If $\Delta \leq \frac{\alpha}{2}$, a robust solution can be found as follows:

Lemma 13. *Let $\sigma = 1$ and $\Delta \leq \frac{\alpha}{2}$. Then $\mathsf{Alg}_{\alpha-\Delta}^+$ is robust. Furthermore, if G is a tree, its price of robustness is $P_{rob}(\mathcal{P}_{\sigma\geq 1}, \mathsf{Alg}_{\alpha-\Delta}^+) \leq 1 + (\alpha - \Delta)/L_{min}$.*

Proof. Let π be a solution computed by $\mathsf{Alg}_{\alpha-\Delta}^+$ and let x be the solution after the recovery phase. Assume that arc $(u, v) \in A$ is delayed by α. Let w_j, $j = 1, \ldots, l$, be the set of nodes directly connected to v by an arc $(v, w_j) \in A$, see Figure 2. We calculate the delays as

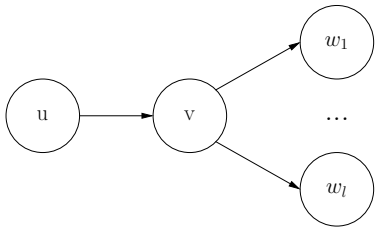

Fig. 2. The DAG for the proof of Lemma 13.

$$x_v - \pi_v = \alpha - (\alpha - \Delta) = \Delta$$
$$x_{w_j} - \pi_{w_j} \leq [\Delta - (\alpha - \Delta)]_+ = 0 \text{ for all } j = 1, \ldots, l,$$

the latter using $\Delta \leq \frac{\alpha}{2}$. Hence,

$$\sum_{u \in V} (x_u - \pi_u) = \Delta.$$

The price of robustness follows from Lemma 11. □

Next, we simplify the network and look at a linear graph. However, this simplification allows to drop the restrictions on Δ and σ from the previous lemmata.

Theorem 5. *Let G be a linear graph. Then Alg_s^+ is robust for $\mathcal{P}_{\sigma\geq 1}$ if and only if*

$$s \geq s^* := \frac{2\sigma\alpha\left(\left\lceil\frac{2\Delta}{\sigma\alpha}\right\rceil+\sigma\right)-\sigma\alpha(\sigma+1)-2\Delta}{\left(\left\lceil\frac{2\Delta}{\sigma\alpha}\right\rceil+\sigma\right)\left(\left\lceil\frac{2\Delta}{\sigma\alpha}\right\rceil+\sigma-1\right)}.$$

Proof. Omitted. □

Corollary 4. *It holds that $s^* \geq \frac{\sigma^2\alpha^2}{2\Delta+\sigma^2\alpha}$ where equality holds if $\frac{s\Delta}{\sigma\alpha}$ is integer.*

Proof. One can compute that $s^* \geq \frac{\sigma^2\alpha^2}{2\Delta+\sigma^2\alpha}$ if and only if

$$\underbrace{\left\lceil\frac{2\Delta}{\sigma\alpha}\right\rceil\sigma\alpha}_{:=A}\underbrace{\left(4\Delta+\sigma\alpha-\sigma\alpha\left\lceil\frac{2\Delta}{\sigma\alpha}\right\rceil\right)}_{:=B} \geq \underbrace{2\Delta}_{:=C}\underbrace{(2\Delta+\sigma\alpha)}_{:=D}.$$

For the latter expression note that $A, B, C, D \geq 0$ and that $A+B = C+D$ and that $A-B \leq D-C$. Hence $AB \geq CD$ and the lower bound is established.

Plugging in s^* in the case that $\frac{s\Delta}{\sigma\alpha}$ is integer shows (after some calculations) that equality holds. □

The price of robustness of algorithm $\mathsf{Alg}_{s^*}^+$ can finally be written down.

Corollary 5. *Let G be a linear graph. Then $P_{rob}(\mathcal{P}_{\sigma\geq 1}, \mathsf{Alg}_{s^*}^+) = 1 + s^*$ where s^* is the minimal slack time of Theorem 5.*

Note that for a concrete scenario, a slack smaller than s^* might also give a robust timetable. This might happen for example if no two source-delayed arcs follow each other or if the size of the network is limited such that at least one node $u \in V$ with a delay of $(x_u - \pi_u) > s^*$ has no outgoing arc. However, we are not interested in one special scenario, but in all possible scenarios from the set of admissible scenarios.

We also remark that this is a discussion of $P_{rob}(\mathcal{P}, \mathsf{Alg}_s^+)$ only. The question if there exists an approach which does better in the worst case is still open. But note that it need not be optimal to add the same slack s to all arcs when the weights w_a are different from each other. This can be seen in the following

Example 2. Consider $G = (V, A)$ with $V = \{v_1, v_2, v_3, v_4\}$, $A = \{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}$, weights $w = (1, 100, 1)$ and lower bounds $L^0 = (1, 1, 1)$, see Fig. 3. Let $\alpha = 4$, $\Delta = 5$ and $\sigma = 1$. If we add the same slack to all arcs, we need at least a slack of 2. With $s = (2, 2, 2)$, we have

$$\sum_{a=(u,v)\in A} w_a(\pi_v - \pi_u) \;=\; \sum_{a=(u,v)\in A} w_a(L_a^0 + s) \;=\; 306$$

(if we schedule each node as early as possible, i.e. $\pi_j := \pi_{j-1} + L_{j-1} + s_{j-1}$, $j = 2, \ldots, 4$). If we allow different slacks on the arcs and set $s = (2, 0, 3)$, we get a robust timetable with

$$\sum_{a=(u,v)\in A} w_a(\pi_v - \pi_u) = 107.$$

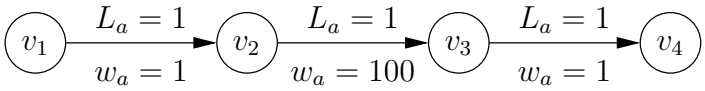

Fig. 3. The DAG for example 2.

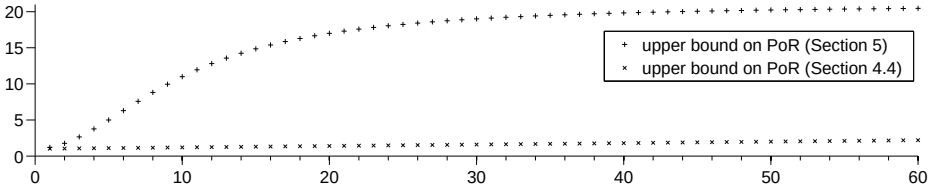

Fig. 4. The price of robustness, depending on the actual restrictions on the recovery algorithm, for $\alpha = 20$ and $\Delta = 1000$ as a function of σ.

6 Conclusions

In this paper, we showed how the concept of recoverable robustness from [15] can be extended to the concept of dynamic recoverable robustness. We showed how this concept can be applied to the delay management problem and suggested different concrete restrictions of the recovery algorithm.

Depending on the concrete restrictions on the recovery algorithms, the price of robustness is very different. In Figure 4, we give the price of robustness for a linear graph if we either restrict the number of nodes being affected by a delay or if we restrict the allowed deviation from the original timetable.

References

1. H. G. Bayer and B. Sendhoff. Robust Optimization - A Comprehensive Survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3190–3218, 2007.

2. A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Mathematical Programming: Special Issue on Robust Optimization*, volume 107. Springer, Berlin, 2006.
3. D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.
4. J.R. Birge and F.V. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, New York, 1997.
5. A. Borodin and R. El-Yaniv, editors. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
6. S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Robust Algorithms and Price of Robustness in Shunting Problems. In *Proc. of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS07)*, 2007.
7. M. Fischetti and M. Monaci. Robust optimization through branch-and-price. In *Proceedings of the 37th Annual Conference of the Italian Operations Research Society (AIRO)*, 2006.
8. M. Fischetti and M. Monaci. Light robustness. Research Paper ARRIVAL-TR-0066, ARRIVAL project, 2008.
9. M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Widmayer. Railway delay management: Exploring its algorithmic complexity. In *Algorithm Theory - Proceedings SWAT 2004*, volume 3111 of *LNCS*, pages 199–211. Springer, 2004.
10. M. Gatto, R. Jacob, L. Peeters, and A. Schöbel. The Computational Complexity of Delay Management. In *Proc. of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 3787 of *Lecture Notes in Computer Science*, 2005.
11. M. Gatto, R. Jacob, L. Peeters, and P. Widmayer. Online Delay Management on a Single Train Line. In *Proc. of the Algorithmic Methods for Railway Optimization (ATMOS04)*, volume 4359 of *Lecture Notes in Computer Science*, 2007.
12. A. Ginkel and A. Schöbel. The bicriteria delay management problem. *Transportation Science*, 41(4):527–538, 2007.
13. L. Giovanni, G. Heilporn, and M. Labbé. Optimization models for the delay management problem in public transportation. *European Journal of Operational Research*, 2006. to appear.
14. P. Kall and S.W. Wallace. *Stochastic Programming*. Wiley, Chichester, 1994.
15. C. Liebchen, M. Lüebbecke, R. H. Möhring, and S. Stiller. Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL Project, 2007.
16. A. Ruszczynski and A. Shapiro, editors. *Stochastic Programming, Handbooks in Operations Research and Management Science Volume 10*. North-Holland, 2003.
17. A. Schöbel. A model for the delay management problem based on mixed integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.
18. A. Schöbel. Book Chapter: Integer programming approaches for solving the delay management problem. volume 4359 of *Lecture Notes in Computer Science*, pages 145–170, 2007.