The Concept of Recoverable Robustness, Linear Programming Recovery, and Railway Applications*

Christian Liebchen, Marco Lübbecke, Rolf Möhring, and Sebastian Stiller

Technische Universität Berlin, Strasse des 17. Juni 136, 10623 Berlin, Germany
{liebchen,m.luebbecke,moehring,stiller}@math.tu-berlin.de

Abstract. We present a new concept for optimization under uncertainty: recoverable robustness. A solution is recovery robust if it can be recovered by limited means in all likely scenarios. Specializing the general concept to linear programming we can show that recoverable robustness combines the flexibility of stochastic programming with the tractability and performances guarantee of the classical robust approach. We exemplify recoverable robustness in delay resistant, periodic and aperiodic timetabling problems, and train platforming.

1 Introduction

Solutions for real-world problems found by mathematical optimization can hardly enter into praxis unless they possess a certain robustness. In applications robustness is not an additional feature but a conditio sine qua non. Usually, robustness is achieved ex post or by rules of thumb, i.e., heuristically. As systems work closer to capacity shortcomings of these suboptimal approaches become apparent. The classical two exact methods to deal with uncertain or fluctuating data in linear programming and combinatorial optimization are (2-Stage) Stochastic Programming and Robust Optimization.

A 2-stage stochastic program is a linear program, where part of the input data are random variables of some distribution. The distribution is either known, or partly known, or can at least be sampled. The decision variables split into *first stage decisions* and *second stage decisions*. The first stage decisions must be chosen fixed for all scenarios. The second stage variables can be chosen after the actual value of the random variables is revealed, i.e., the second stage decision can be different for each scenario. Thus, strictly speaking the second stage variables form a vector of random variables. But it is natural to interpret this vector of random variables as a large (deterministic) vector containing for each scenario one copy of the second stage variables. To be feasible in each scenario first and second stage variables together must form a feasible vector for the data realized in the scenario. Usually, the objective function of a 2-stage stochastic program

* This work was partially supported by the Future and Emerging Technologies Unit of EC, under contract no. FP6-021235-2 (FP6 IST/FET Open/Project ARRIVAL).

R.K. Ahuja et al. (Eds.): Robust and Online Large-Scale Optimization, LNCS 5868, pp. 1–27, 2009.

comprises a cost function for the first stage variables and the *expected* cost of the second stage variables according to the given distribution. Assuming a discretized scenario set there is an obvious way to interpret a 2-stage stochastic linear program as a (very large) usual linear program: The random variables in the original linear program are resolved by adding for each scenario a copy of the original linear program. In this copy the random variable is replaced by its realization in the scenario. This is called the scenario expansion of a 2-stage stochastic program.

Classical robust optimization considers a quite similar situation, except that one abstains from second stage actions. Again for a linear program a certain part of the data is subject to uncertainty. But as the robust program features no second stage variables, the variables—which are fixed before the actual data is revealed—must form a feasible solution for *every* scenario. A robust solution fits for all scenarios. Likewise, the objective function of a robust program contains no expectation or other stochastic component. The objective is a deterministic, linear function of the solution. Obviously, a robust model avoids the use of probability distributions. It suffices to know the range, in which the uncertain data can fluctuate. Usually, one models this range smaller than given in reality, thus excluding extremely unlikely scenarios.

Both methods have their merits for different types of applications. Still, for a number of applications none of the two provides a suitable method. One of these applications is delay resistant timetabling, e.g., for a railway system. Here instances are usually too large for stochastic programming approaches. Whereas robust optimization appeals for two reasons: A robust solution comes with a guarantee to be feasible in all scenarios of a certain restricted scenario set. We call this the set of *likely scenarios*. In addition, robust optimization yields compact mathematical models which are likely to be solvable on a scale relevant for practical purposes like delay resistant timetabling. But it turns out that the classical robust approach [1,15]—which we call *strict robustness* and which is a special case of the broader concept we present here—is necessarily over-conservative in the context of timetabling. The strict robust model yields a timetable where *each* train ride is scheduled to take its technically minimal travel time plus at least the total time of disturbances that is likely in the *whole* network. Unfortunately, this over-conservativism is intrinsic to the classical robust approach.

In practice, one often encounters a way of handling disturbances that is different from both methods mentioned above. First, the plan is furnished with some slack that shall allow to compensate disturbances. Second, the plan can be recovered during operations. Third, these recoveries are limited. The limits apply to the actions that can be taken and the computational means by which the recovery is calculated. For example, changes to the plan may be restricted to be local or to only affect a certain subset of the planning variables. It is a promising, practical concept that the plan has to be recoverable by limited means in every likely scenario. Still, in practice the plans, their slack and the simple means of recovery are currently not optimized together.

Related work. The concept of *recoverable robustness* has first been formalized together with concepts for the price of robustness in [11]. In the meantime it has attracted several applications to optimization problems particularly in the railway context. In [5,7] the concept has successfully been applied to shunting problems. Specific cases of recoverable robust timetabling are treated in [4,6,7]. In particular, they can identify some types of scenario sets for which finding a recoverable robust timetable becomes NP-hard and other types of scenario sets for which an efficient algorithm exists. In [9] the concept of recoverable robustness is spelled out specifically for the case of multi-stage recovery. Dynamic algorithms for this case have been proposed in [8]. In case the recovery is a linear program [16] provides for efficient algorithms and a stochastic analysis of quality for recovery robust solutions both for the case of right-hand side disturbances and for the case of matrix disturbances. The first application of recoverable robustness in a study on real-world data was carried out in [3]. It uses the techniques developed in this work and will be described in some detail in Section 5.

Our contribution and Outline. In this work we present the concept of *Recoverable Robustness*. The goal of recoverable robustness is to *jointly optimize* the plan and the strategy for limited recovery. This will combine the flexibility of stochastic programming with the performance guarantee and the compactness of models found in robust optimization.

In Section 2 we develop the concept of recoverable robustness formally and in full generality. As delay resistant timetabling has sparked its development, we exemplify the modeling power of recoverable robustness by the case of timetabling in Section 3. To solve recovery robust models we specify to *Linear Recovery Robust Programs* in Section 4, for which we provide an efficient algorithm. Finally, we demonstrate the power of this method by citing a real-world application of the method to the train platforming problem (Section 5).

2 The Concept of Recoverable Robustness

We are looking for solutions to an optimization problem which in a limited set of scenarios can be made feasible, or *recovered*, by a limited effort. Therefore, we need to define

- the **o**riginal optimization problem (Step O),
- the imperfection of information, that is, the **s**cenarios (Step S), and
- the limited **r**ecovery possibilities (Step R).

For Step O and Step S a large toolbox for modeling can be borrowed from classical approaches to optimization respectively optimization with imperfect information. Step R is a little less obvious, and we choose to formalize it via a *class $\mathcal{A}$ of admissible recovery algorithms.*

A solution x for the optimization problem defined in Step O is *recovery-robust*

- *against* the imperfection of information (Step S) and
- *for* the recovery possibilities (Step R),

if in all situations that may occur according to Step S, we can recover from x a feasible solution by means of one of the algorithms given in Step R.

Computations in recovery-robust optimization naturally decompose into a *planning phase* and a *recovery phase*. In the planning phase,

- we compute a solution x which may become infeasible in the realized scenario,
- and we choose $A \in \mathcal{A}$, i.e., one of the admissible recovery algorithms.

Such a pair (x, A) hedges for data uncertainty in the sense that in the recovery phase

- algorithm A is used to turn x into a feasible solution in the realized scenario.

The output (x, A) of the planning phase is more than a solution, it is a *precaution*. It does not only state that recovery is possible for x, but explicitly specifies *how* this recovery can be found, namely by the algorithm A.

The formal definition of recoverable robustness [11] we give next is very broad. The theorems in this paper only apply to strong specializations of that concept.

We introduce some terminology. Let $\mathcal{F}$ denote the original optimization problem. An instance $O = (P, f)$ of $\mathcal{F}$ consists of a set P of feasible solutions, and an objective function $f : P \to \mathbb{R}$ which is to be minimized.

By $\mathcal{R} = \mathcal{R}_{\mathcal{F}}$ we denote a model of imperfect information for $\mathcal{F}$ in the sense that for every instance O we specify a set $S = S_O \in \mathcal{R}_{\mathcal{F}}$ of possible scenarios. Let P_s denote the set of feasible solutions in scenario $s \in S$.

We denote by $\mathcal{A}$ a class of algorithms called *admissible recovery algorithms*. A recovery algorithm $A \in \mathcal{A}$ solves the *recovery problem*, which is a feasibility problem. Its input is $x \in P$ and $s \in S$. In case of a *feasible recovery*, $A(x, s) \in P_s$.

Definition 1. *The triple* $(\mathcal{F}, \mathcal{R}, \mathcal{A})$ *is called a* recovery robust optimization problem*, abbreviated RROP.*

Definition 2. *A pair* $(x, A) \in P \times \mathcal{A}$ *consisting of a* planning solution x *and an* admissible algorithm A *is called a* precaution.

Definition 3. *A precaution is* recovery robust*, iff for every scenario* $s \in S$ *the recovery algorithm* A *finds a feasible solution to the recovery problem, i.e., for all* $s \in S$ *we have* $A(x, s) \in P_s$.

Definition 4. *An* optimal precaution *is a recovery robust precaution* (x, A) *for which* $f(x)$ *is minimal.*

Thus, we can quite compactly write an RROP instance as

$$\inf_{(x,A)\in P\times\mathcal{A}} f(x)$$
$$\text{s.t.} \quad \forall s \in S : A(x,s) \in P_s \quad .$$

The objective function value of an RROP is infinity, if no recovery is possible for some scenario with the algorithms given in the class $\mathcal{A}$ of admissible recovery algorithms.

It is a distinguished feature of this notion that the planning solution is explicitly accompanied by the recovery algorithm. In some specializations the choice of the algorithm is self-understood. For example, for linear recovery robust programs, to which we will devote our main attention, the algorithm is some solver of a linear program or a simpler algorithm that solves the specific type of linear program that arises as the recovery problem of the specific RROP. Then we will simply speak of the planning solution x, tacitly combining it with the obvious algorithm to form a precaution.

2.1 Restricting the Recovery Algorithms

The class of admissible recovery algorithms serves as a very broad wildcard for different modeling intentions. Here we summarize some important types of restrictions that can be expressed by means of that class.

The definition of the algorithm class $\mathcal{A}$ also determines the computational balance between the planning and the recovery phase. For all practical purposes, one must impose sensible limits on the recovery algorithms (otherwise, the entire original optimization problem could be solved in the recovery phase, when the realized scenario is known). In very bold term, these limits fall into two categories:

- limits on the actions of recovery;
- limits on the computational power to find those actions of recovery.

We mention two important subclasses of the first category:

Strict Robustness. We can forbid recovery entirely by letting $\mathcal{A}$ consist of the single recovery algorithm A with $A(x,s) = x$ for all $s \in S$. This is called *strict robustness*. Note that by strict robustness the classical notion of robust programming is contained in the definition of recoverable robustness.

Recovery Close to Planning. An important type of restrictions for the class of admissible recovery algorithms is, that the recovery solution $A(x,s)$ must not deviate too far from the original solution x according to some measure of distance defined for the specific problem. For some distance measures one can define subsets $P_{s,x} \subseteq P_s$ depending on the scenario s and the original solution x, such that the restriction to the recovery algorithm that $A(x,s)$ will not deviate too far from x, can be expressed equivalently by requiring $A(x,s) \in P_{s,x}$. As

an example, think of a railway timetable that must be recovered, such that the difference between the actual and the planned arrival times is not too big, i.e., that the delay is limited.

2.2 Passing Information to the Recovery

If (as it ought to be) the recovery algorithms in $\mathcal{A}$ are allowed substantially less computational power than the precaution algorithms in $\mathcal{B}$, we may want to pass some additional information $z \in Z$ (for some set Z) about the instance to the recovery algorithm. That is, we may compute an extended precaution $B(P, f, S) = (x, A, z)$, and in the recovery phase we require $A(x, s, z) \in P_s$.

As a simple example, consider a class of admissible recovery algorithms $\mathcal{A}$ that is restricted to computational effort linear in the size of a certain finite set of weights, which is part of the input of the RROP instance. Then it might be helpful to pass an ordered list of those weights on to the recovery algorithm, because the recovery algorithm will not have the means to calculate the ordered list itself, but could make use of it.

In Section 3 we present another example, namely rule based delay management policies, which shows that it is a perfectly natural idea to preprocess some values depending on the instance, with which the recovery algorithm becomes a very simple procedure.

2.3 Limited Recovery Cost

The recovery algorithm A solves a feasibility problem, and we did not consider any cost incurred by the recovery so far. There are at least two ways to do so in the framework of recoverable robustness. Let $d(y^s)$ be some (possibly vector valued) function measuring the cost of recovery $y^s := A(x, s)$.

- **Fixed Limit:** Impose a fixed limit λ to $d(y^s)$ for all scenarios s.
- **Planned Limit:** Let λ be a (vector of) variable(s) and part of the planning solution. Require $\lambda \geq d(y^s)$ for every scenario s, and let $\lambda \in \Lambda$ influence the objective function by some function $g : \Lambda \to \mathbb{R}$.

In the second setting, the planned limit λ to the cost of recovery is a variable chosen in the planning phase and then passed to the recovery algorithm A. It is the task of A to respect the constraint $\lambda \geq d(y)$, and it is the task of the planning phase to choose (x, A, λ), such that A will find a recovery for x with cost less or equal to λ. Therefore, and to be consistent with previous notation we formulate the cost bound slightly different. Let P'_s denote the set of feasible recoveries for scenario s. Then we define P_s by:

$$A(x, s, \lambda) \in P_s :\Leftrightarrow d(A(x, s)) \leq \lambda \wedge A(x, s) \in P'_s$$

We obtain the following recovery robust optimization problem with recovery cost:

$$\min_{(x,A,\lambda)\in P\times\mathcal{A}\times\Lambda} f(x)+g(\lambda)$$
$$\text{s.t.} \quad \forall s\in S: A(x,s,\lambda)\in P_s \quad .$$

Including the possibility to pass some extra information $y \in Y$ to A we obtain:

$$\min_{(x,A,z,\lambda)\in P\times\mathcal{A}\times Z\times\Lambda} f(x)+g(\lambda)$$
$$\text{s.t.} \quad \forall s\in S: A(x,s,z,\lambda)\in P_s \quad .$$

These recovery cost aware variants allow for computing an optimal trade-off between higher flexibility for recovery by a looser upper bound on the recovery cost, against higher cost in the planning phase. This is conceptually close to two-stage stochastic programming, however, we do not calculate an expectation of the second stage cost, but adjust a common upper bound on the recovery cost. This type of problem still has a purely deterministic objective. The linear recovery robust programs discussed later are an example of this type of RROP.

3 Recovery Robust Timetabling

Punctual trains are probably the first thing a layman will expect from robustness in railways. Reliable technology and well trained staff highly contribute to increased punctuality. Nevertheless, modern railway systems still feature small disturbances in every-day operations.

A typical example for a disturbance is a prolonged stop at a station because of a jammed door. A disturbance is a seminal event in the sense that the disturbance may cause several delays in the system but is not itself caused by other delays. Informing passengers about the reason for a delay affecting them, railway service providers sometimes do not distinguish between disturbances, i.e., seminal events, and delays that are themselves consequences of some initial disturbance. We will use the term *disturbance* exclusively for initial changes of planning data. A *delay* is any difference between the planned point in time for an event and the time the event actually takes place. We also speak of *negative delay*, when an event takes place earlier than planned.

A good timetable is furnished with *buffers* to absorb small disturbances, such that they do not affect the planned arrival times at all, or that they cause only few delays in the whole system. Those buffer times come at the expense of longer, planned travel times. Hence they must not be introduced excessively. Delay resistant timetabling is about increasing the planned travel times as little as possible, while guaranteeing the consequences of small disturbances to be limited.

We will now show how delay resistant timetabling can be formulated as a recovery robust optimization problem. We actually show that a robust version

of timetabling is only reasonable, if it is understood as a recovery robust optimization problem. Moreover, we show how recoverable robustness integrates timetabling and the so-called delay management. Delay management is the term coined for the set of operational decisions reacting to concrete disturbances, i.e., the recovery actions. Its integration with timetabling is an important step forward for delay resistant timetabling, which can be formalized by the notion of recoverable robustness.

Step O. The original problem is the deterministic timetabling problem. It exists in many versions that differ in the level of modeling detail, the objective function, or whether periodic or aperiodic plans are desired. The virtues of recovery robust timetables can already be shown for a simple version.

A Simple Timetabling Problem. The basic mathematical model that stands to reason for timetabling problems is the so-called Event-Activity Model or Feasible Differential Problem [13]. A timetable assigns points in time for certain events, i.e., arrivals and departures of trains. This assignment is feasible, if the differences in time between two related events are large enough, to allow for the activities relating them. For example, the arrival of a train must be scheduled sufficiently after its departure at the previous station. Likewise, transfers of passengers require the arrival of the feeder train and the departure of the transfer train to take place in the right time order and with a time difference at least large enough to allow for the passengers to change trains. We now describe a basic version of this model.

The input for our version of timetabling is a directed graph $G = (V, E)$ together with a non-negative function $t : E \to \mathbb{R}_+$ on the arc set. The nodes of the graph $V = V_{\mathrm{AR}} \cup V_{\mathrm{DP}}$ model arrival events (V_{AR}) and departure events (V_{DP}) of trains at stations. The arc set can be partitioned into three sets representing traveling of a train from one station to the next, E_{DR}, stopping of a train at a station, E_{ST}, and transfers of passengers from one train to another at the same station, E_{TF}. For travel arcs $e = (i, j) \in E_{\mathrm{DR}}$ we have $i \in V_{\mathrm{DP}}$ and $j \in V_{\mathrm{AR}}$, for the two other types $e = (i, j) \in E_{\mathrm{ST}} \cup E_{\mathrm{TF}}$ the contrary holds: $i \in V_{\mathrm{AR}}$ and $j \in V_{\mathrm{DP}}$. The function $t(e)$ expresses the minimum time required for the action corresponding to $e = (i, j)$, in other words the minimum time between event i and event j. For example, for a travel arc e the value of the function $t(e)$ expresses the technical travel time between the two stations.

A feasible timetable is a non-negative vector $\pi \in \mathbb{R}_+^{|V|}$ such that $t(e) \leq \pi_j - \pi_i$ for all $e = (i, j) \in E$. W.l.o.g. we can assume that G is acyclic.

For the objective function we are given a non-negative weight function $w : E \to \mathbb{R}_+$, where $w_e = w(e)$ states how many passengers travel along arc e, i.e., are in the train during the execution of that action, or change trains according to that transfer arc. An optimal timetable is a feasible timetable that minimizes the total planned (or *nominal*) travel time of the passengers:

$$\sum_{e=(i,j)\in E} w_e(\pi_j - \pi_i).$$

Thus the data for the original problem can be encoded in a triple (G, t, w), containing the event-activity graph G, the arc length function t, and a cost function on the arcs w. The original problem can formulated as a linear program:

$$\min \sum_{e=(i,j)\in E} w_a(\pi_j - \pi_i)$$

$$\text{s.t.} \quad \pi_j - \pi_i \geq t(e) \quad \forall e = (i,j) \in E$$

$$\pi \geq 0$$

Step S. We assume uncertainty in the time needed for traveling and stopping. Those actions typically produce small disturbances. For a scenario s we are given a function $t^s : E \to \mathbb{R}_+$, with the properties $t^s(e) \geq t(e)$ for all $e \in E$, and $t^s(e) = t(e)$ for all $e \in E_{\text{TF}}$. As we only want to consider scenarios with small disturbances, we restrict to those scenarios where $t^s(e) - t(e) \leq \Delta_e$, for some small, scenario independent constant Δ_e. In a linear program one can scale each row, i.e., multiply all matrix entries of the row and the corresponding component of the right-hand side vector by a positive scalar, without changing the set of feasible solutions. Therefore, we can assume w.l.o.g. $\Delta_e = \Delta$ for all $e \in E$. Additionally, we require that not too many disturbances occur at the same time, i.e., in every scenario for all but k arcs $e \in E$ we have $t^s(e) = t(e)$.

Of course, there are situations in practice where larger disturbances occur. But it is not reasonable to prepare for such catastrophic events in the published timetable.

Strict Robustness. The above restrictions to the scenario set can be very strong, in particular, if we choose $k = 1$. But even for such a strongly limited scenario set strict robustness leads to unacceptably conservative timetables. Namely, the strict robust problem can be formulated as the following linear program:

$$\min \sum_{e=(i,j)\in E} w_e(\pi_j - \pi_i)$$

$$\text{s.t.} \quad \pi_j - \pi_i \geq t(e) + \Delta \quad \forall e = (i,j) \in E$$

$$\pi \geq 0$$

In other words, even if we assume that in every scenario at most one arc takes Δ time units longer, we have to construct a timetable as if all (traveling and stopping) arcs were Δ time units longer. This phenomenon yields solutions so conservative, that classical robust programming is ruled out for timetabling. Indeed, delay resistant timetabling has so far been addressed by stochastic programming [12,17] only. These approaches suffer from strong limitations to the size of solvable problems.

The real world expectation towards delay resistant timetables includes that the timetable can be adjusted slightly during its operation. But a strict robust program looks for timetables that can be operated unchanged despite disturbances. This makes the plans too conservative even for very restricted scenario sets. Robust timetabling is naturally *recovery robust timetabling* as we defined it. Naturally, a railway timetable has to be robust against *small* disturbances and for *limited* recovery.

Step R. The recovery of a timetable is called *delay management*. The two central means of delay management are delaying events and canceling transfers. Delaying an event means to propagate the delay through the network. Canceling a transfer means to inhibit this propagation at the expense of some passengers loosing their connection.

Pure delay propagation seems not deserve the name recovery at all. But recall that if delay propagation is not captured in the model, as in the strict robust model, the solutions become necessarily over-conservative. Delay is a form of recovery, and though it is a basic, it is a very important.

Actually, delay management has several other possibilities for recovery. For example, one may cancel train trips, re-route the trains, or re-route the passengers by advising them to use an alternative connection, or hope that they will figure such a possibility themselves. Moreover, delay management has to pay respect to several other aspects of the transportation system. For example, the shifts of the on-board crews are affected by delays. These in turn may be subject to subtile regulations by law or contracts and general terms of employment.

We initially adopt a quite simple perspective to delay management gradually increasing the complexity of the model. First we concentrate on delay, later on delay and broken transfers. The latter means plan with respect to delay management decisions, i.e., decision whether a train shall wait for delayed transferring passengers, or not in order to remain itself on time. Even basic delay management decision lead to PSPACE-hard models.

Roughly speaking, for a PSPACE-hard problem we cannot even recognize an optimal solution, when it is given to us, nor can we compare two solutions suggested to us. (See below for details on the complexity class PSPACE.) Therefore, we describe a variant that yields simpler models and is useful in railway practice.

Simple Recovery Robust Timetabling. First, we describe a model where the recovery can only delay the events but cannot cancel transfers. This is not a recovery in the ordinary understanding of the word. The recovery is simply the delay propagation. But this simple recovery already rids us from the conservatism trap of strict robustness.

In the recovery phase, when the scenario s and its actual traveling and stopping times t^s are known, we construct a *disposition timetable* $\pi^s \in \mathbb{R}_+^{|V|}$ fulfilling the following feasibility condition:

– The disposition timetable π^s of scenario s must be feasible for t^s, i.e.,

$$\forall e = (i, j) \in E : \pi_j^s - \pi_i^s \geq t^s(e).$$

These inequalities define the set (actually, the polytope) P_s of feasible recoveries in scenario s.

If this was the complete set of restrictions to the recovery, every timetable would be recoverable. We set up limits to the recovery algorithms:

TTC. The disposition timetable is bounded by the original timetable in a very strict manner: Trains must not depart earlier than scheduled, i.e.,

$$\forall e \in E_{\mathrm{DP}} : \pi^s(e) \geq \pi(e).$$

This is what we call the *timetabling condition*.

L1. We want the sum of the delays of all arrival events to be limited. Therefore assume we are also given a weight function $\ell : V_{\mathrm{AR}} \to \mathbb{R}_+$ that states how many passengers reach their final destination by the arrival event i. We fix a limit $\lambda_1 \geq 0$ and require:

$$\sum_{i \in V_{\mathrm{AR}}} \ell(i)\,(\pi_i^s - \pi_i) \leq \lambda_1.$$

L2. One may additionally want to limit the delay for each arrival separately, ensuring that no passenger will experience an extreme delay exceeding some fixed $\lambda_2 \geq 0$, i.e.:

$$\forall i \in V_{\mathrm{AR}} : \pi_i^s - \pi_i \leq \lambda_2.$$

In our model a recovery algorithm $A \in \mathcal{A}$ must respect all three limits. The bounds λ_1 and λ_2 can be fixed a priori, or made part of the objective function. In this way upper bounds on the *recovery cost* can be incorporated into the optimization process. For a timetabling problem (G, t, w) and a function $\ell : V_{\mathrm{AR}} \to \mathbb{R}_+$ and constants $g_1, g_2 \geq 0$ and an integer k we can describe the first timetabling RROP by the following linear program:

$$\min \sum_{e=(i,j)\in E} w_e(\pi_j - \pi_i) + g_1 \cdot \lambda_1 + g_2 \cdot \lambda_2$$

$$\begin{aligned}
\text{s.t.} \quad \pi_j - \pi_i &\geq t(e) \quad \forall e = (i,j) \in E && (1)\\
\pi_j^s - \pi_i^s &\geq t^s(e) \quad \forall s \in S, \forall e = (i,j) \in E && (2)\\
\pi_i^s &\geq \pi_i \quad \forall s \in S, \forall i \in V_{\mathrm{DP}} && (3)\\
\sum_{i \in V_{\mathrm{AR}}} \ell(i)\,(\pi_i^s - \pi_i) &\leq \lambda_1 \quad \forall s \in S && (4)\\
\pi_i^s - \pi_i &\leq \lambda_2 \quad \forall s \in S, \forall i \in V_{\mathrm{AR}} && (5)\\
\lambda_{\{1,2\}}, \pi^s, \pi &\geq 0
\end{aligned}$$

The set of scenarios S in this description is defined via the set of all functions $t^s : E \to \mathbb{R}_+$ which fulfill the following four conditions from Step S:

$$t^s(e) \geq t(e) \quad \forall e \in E$$

$$t^s(e) \leq t(e) + \Delta \quad \forall e \in E$$
$$t^s(e) = t(e) \quad \forall e \in E_{\mathrm{TF}}$$
$$|\{e \in E : t^s(e) \neq t(e)\}| \leq k$$

In our terminology Inequality (1) defines P, Inequality (2) defines P_s, Inequalities (3) to (5) express limits to the action of the algorithm, namely, that the recovery may not deviate to much from the original solution. In detail (3) models the TTC, (4) ensures condition L1 and (5) condition L2.

Here and in the remainder of the example we use mathematical programs to *express* concisely the problems under consideration. These programs are not necessarily the right approach to *solve* the problems. Note that the above linear program is a scenario expansion and therefore too large to be solved for instances of relevant scale. In Section 4, we will devise a general result that allows us to reformulate such scenario expansions in a compact way. Thereby, the recovery robust timetabling problem becomes efficiently solvable.

Breaking Connections. In practice delay management allows for a second kind of recovery. It is possible to cancel transfers in order to stop the propagation of delay through the network. We now include the possibility to cancel transfers into the recovery of our model.

Again we consider a simple version for explanatory purposes. A transfer arc e can be removed from the graph G at a fixed cost $g_3 \geq 0$ multiplied with the weight w_e. With a sufficiently large constant M we obtain a mixed integer linear program representing this model:

$$\min \sum_{e=(i,j)\in E} w_e(\pi_j - \pi_i) + g_1 \cdot \lambda_1 + g_2 \cdot \lambda_2 + g_3 \cdot \lambda_3$$

$$\text{s.t.} \quad \pi_j - \pi_i \geq t(e) \quad \forall e = (i,j) \in E \tag{6}$$
$$\pi_j^s - \pi_i^s \geq t^s(e) \quad \forall s \in S, \forall e = (i,j) \in E_{\mathrm{DR}} \cup E_{\mathrm{ST}} \tag{7}$$
$$\pi_j^s - \pi_i^s + M x_e^s \geq t^s(e) \quad \forall s \in S, \forall e = (i,j) \in E_{\mathrm{TF}} \tag{8}$$
$$\pi_i^s \geq \pi_i \quad \forall s \in S, \forall i \in V_{\mathrm{DP}} \tag{9}$$
$$\sum_{i \in V_{\mathrm{AR}}} \ell(i)\,(\pi_i^s - \pi_i) \leq \lambda_1 \quad \forall s \in S \tag{10}$$
$$\pi_i^s - \pi_i \leq \lambda_2 \quad \forall s \in S, \forall i \in V_{\mathrm{AR}} \tag{11}$$
$$\sum_{e \in E_{\mathrm{TF}}} w_e x_e^s \leq \lambda_3 \quad \forall s \in S \tag{12}$$

$$\lambda_{\{1,2,3\}}, \pi^s, \pi \geq 0$$
$$x^s \in \{0,1\}^{|E_{\mathrm{TF}}|}$$

In our terminology Inequality 6 defines P. Inequalities 7 and 8 define P_s for every s. Again Inequalities 9 to 11 express limits to the actions that can be taken by the

recovery algorithms. These are limits to the deviation of the recovered solution π^s from the original solution π.

3.1 Computationally Limited Recovery Algorithms

So far we imposed limits on the *actions* of the recovery algorithms. But delay management is a real-time task. Decisions must be taken in very short time. Thus it makes sense to impose further restrictions on the computational power of the recovery algorithm. Note that in general such restrictions cannot be expressed by a mathematical program as above. We now give two examples for computationally restricted classes of recovery algorithms.

The Online Character of Delay Management. In fact the above model has a fundamental weakness. It assumes that the recovered solution, i.e., the disposition timetable $\pi^s = A(\pi, s)$ can be chosen after s is known completely. This is of course not the case for real-world delay management: The disturbances evolve over time, and delay management must take decisions before the whole scenario is known. This means that the algorithms in $\mathcal{A}$ must be non-anticipative[1].

PSPACE-hardness of Delay Management. The multistage structure of some delay management models, namely that uncertain events and dispatching decisions alternate, makes these problems extraordinarily hard. Even quite restricted models have been shown to be PSPACE-hard [2].

The complexity class PSPACE contains those decision problems that can be decided with the use of memory space limited by a polynomial in the input size. The class NP is contained in PSPACE, because in polynomial time only polynomial space can be used. It is widely assumed that NP is a proper subset of PSPACE. Given this, one cannot decide in polynomial time that a given solution to a PSPACE-hard problem is feasible, because else the solution would be a certificate and therefore the problem in NP. (Note, that the complexity terminology is formulated for decision problems. Feasibility in this context means, that the delay management solution is feasible in the usual sense and in addition has cost less or equal to some constant.) Thereby it becomes even difficult to assess the quality of a solution for delay management, or to compare the quality of two competing delay management strategies.

Rule Based Delay Management. The previous observation is quite discouraging. How shall one design a recovery robust timetable, if the recovery itself is already PSPACE-hard? We now dicuss a special restriction to the delay management

[1] Given a mapping of the random variables in the input and of the decision variables to some partially ordered set (i.e., a timeline). Then a (deterministic) algorithm is *non-anticipative*, if for any pair of scenarios s and s' and every decision variable x, the algorithm in both scenarios chooses the same value for x, whenever s and s' are equal in all data entries that are mapped to elements less or equal to the image of x. This means that the algorithm can at no time anticipate and react to data revealed later.

that is motivated by the real-world railway application and turns each decision whether to wait or not to wait into a constant time solvable question. The model keeps the multistage character, but the resulting recovery robust timetabling problem is solvable by a mixed integer program.

Delay management decisions must be taken very quickly. Moreover, as delay management is a very sensitive topic for passengers' satisfaction the transparency of delay management decisions can be very valuable. A passenger might be more willing to accept a decision, that is based on explicit rules, about how long, e.g., a local train waits for a high-speed train, than to accept the outcome of some non-transparent heuristic or optimization procedure. For these two reasons, computational limits for real-time decisions and transparency for the passenger, one may want to restrict the class $\mathcal{A}$ of admissible recovery algorithms to *rule based delay management*. The idea is that trains will wait for at most a certain time for the trains connecting to them. These maximal waiting times depend on the type of involved trains. For example, a local train might wait 10 minutes for a high-speed train, but vice versa the waiting time could be zero. Fixing the maximal waiting times determines the delay management (within the assumed modeling precision). But, which waiting times are best? Does the asymmetry in the example make sense? We want to optimize the waiting rules, i.e., the delay management together with the timetable.

Assume we distinguish between m types of trains in the system, i.e., we have a mapping $\mu : V \to \{1, \dots, m\}$ of the events onto the train types. A rule based delay management policy A is specified by a matrix $\mathcal{M} = \mathcal{M}_A \in \mathbb{R}_+^{m \times m}$. The y-th entry in the x-th row m_{xy} is the maximum time a departure event of train type y will be postponed in order to ensure transfer from a type x train. Formally, a rule based delay management policy schedules a departure event j at the earliest time π_j^s satisfying

$$\begin{aligned} \pi_j^s &\geq \pi_i^s + t^s(i,j) \quad \forall (i,j) \in E_{\mathrm{ST}} \\ \pi_j^s &\geq \min\{\pi_i^s + t^s(i,j), \pi_j + m_{\mu(i)\mu(j)}\} \quad \forall (i,j) \in E_{\mathrm{TF}} \\ \pi_j^s &\geq \pi_j \,. \end{aligned}$$

Arrival events are scheduled as early as possible respecting TTC and the traveling times in scenario s:

$$\pi_j^s = \max(\{\pi_j\} \cup \{\pi_i^s + t^s(i,j) | (i,j) \in E_{\mathrm{DR}}\}) \quad \forall j \in V_{\mathrm{AR}}$$

Actually, the maximum is taken over two elements, as only one traveling arc (i,j) leads to each arrival event j.

Moreover, for a transfer arc $(i,j) \in E_{\mathrm{TF}}$ the canceling variable $x_{(i,j)}$ is set to 1 if and only if the result of the above rule gives $\pi_i^s + t(i,j) > \pi_j^s$.

It is easy to see that such a recovery algorithm gives a feasible recovery for every (even non-restricted) scenario s and every solution $\pi \in P$. If we restrict $\mathcal{A}$ to the class of rule based delay management policies, the RROP consists in finding a $m \times m$ matrix $\mathcal{M}$ and a schedule π that minimizes an objective function like those in the models we presented earlier:

$$\min_{\mathcal{M},\pi,\lambda_{\{1,2,3\}}} \sum_{e=(i,j)\in E} w_e(\pi_j - \pi_i) + g_1 \cdot \lambda_1 + g_2 \cdot \lambda_2 + g_3 \cdot \lambda_3$$

$$\text{s.t.} \quad \pi_j - \pi_i \geq t(e) \quad \forall e = (i,j) \in E \tag{13}$$

$$\forall s \in S, \forall j \in V_{\text{DP}}: \tag{14}$$

$$\begin{aligned} \pi_j^s = \max(&\{\pi_i^s + t^s(i,j) \mid (i,j) \in E_{\text{ST}}\} \\ &\cup \max\{\min\{\pi_i^s + t^s(i,j), \pi_j + m_{\mu(i)\mu(j)}\} \mid (i,j) \in E_{\text{TF}}\} \\ &\cup \{\pi_j\}) \end{aligned}$$

$$\forall s \in S, \forall j \in V_{\text{AR}}: \tag{15}$$

$$\begin{aligned} \pi_j^s = \max(&\{\pi_j\} \\ &\cup \{\pi_i^s + t^s(i,j) \mid (i,j) \in E_{\text{DR}}\} \end{aligned}$$

$$\pi_j^s - \pi_i^s + M x_e^s \geq t^s(e) \quad \forall s \in S, \forall e = (i,j) \in E_{\text{TF}} \tag{16}$$

$$\sum_{i \in V_{\text{AR}}} \ell(i)\,(\pi_i^s - \pi_i) \leq \lambda_1 \quad \forall s \in S \tag{17}$$

$$\pi_i^s - \pi_i \leq \lambda_2 \quad \forall s \in S, \forall i \in V_{\text{AR}} \tag{18}$$

$$\sum_{e \in E_{\text{TF}}} w_e x_e^s \leq \lambda_3 \quad \forall s \in S \tag{19}$$

$$\lambda_{\{1,2,3\}}, \pi^s, \pi \geq 0$$

$$x^s \in \{0,1\}^{|E_{\text{TF}}|}$$

The timetabling condition is ensured automatically by the rule based delay management described in Equations (14) and (15).

Rule based delay management algorithms are non-anticipative. The formulation we give even enforces the following behavior: The departure π_i of a train A will be delayed for transferring passengers from train B (with arrival π_j) for the maximal waiting time $m_{\mu(i)\mu(j)}$, even if before time $\pi_i + m_{\mu(i)\mu(j)}$ it becomes known that train B will arrive too late for its passengers to reach train A at time $\pi_i + m_{\mu(i)\mu(j)}$. As formulated, a train will wait the due time, even if the awaited train is hopelessly delayed. In practice, delay managers might handle such a situation a little less short minded.

Rule based delay management is a good example for the idea of integrating robust planning and simple recovery. Consider the following example of two local trains, A and B, and one high-speed train C. Passengers transfer from A to B, and from B to C. Assume local trains wait 7 minutes for each other, but high-speed trains wait at most 2 minutes for local trains. Then train A being late could force train B to loose its important connection to the high-speed train C. Indeed, this could happen, if the timetable and the waiting times are not attuned. In the planning, we might not be willing to increase the time a high-speed train waits, but instead plan a sufficient buffer for the transfer from B to C. This example illustrates that buffer times and waiting rules must be constructed jointly in order to attain optimal delay resistance.

4 Linear Programming Recovery

In this section we specialize to RROPs linear programs as recovery. We call such an RROP a Linear Recovery Robust Problem (LRP). We show how LRPs can be solved for certain scenario sets. This leads us to a special case, namely robust network buffering, which entails the robust timetabling problem. Towards the end of this section we turn to a variant of LRPs, where the planning problem is an *integer* linear program.

4.1 Linear Recovery Robust Programs

Given a linear program ($\min c'x$, s.t. $A^0x \geq b^0$) with m rows and n variables. We seek solutions to this problem that can be recovered by limited means in a certain limited set of disturbance scenarios. The situation in a disturbance scenario s is described by a set of linear inequalities, notably, by a matrix A^s and a right-hand side b^s. We slightly abuse notation when we say that the scenario set S contains a scenario (A^s, b^s), which, strictly speaking, is the image of scenario s under the random variable (A, b). We will discuss later more precisely the scenario sets considered in this analysis. For the linear programming case the limited possibility to recover is defined via a recovery matrix $\hat{A}$, a recovery cost d, and a recovery budget D. A vector x is recovery robust, if for all (A^s, b^s) in the scenario set S exists y such that $A^sx + \hat{A}y \geq b^s$, and $d'y \leq D$. Further, we require that x is feasible for the original problem without recovery, i.e., $A^0x \geq b^0$. The problem then reads:

$$\begin{aligned} \inf_x \; & c'x \\ \text{s.t.} \quad & A^0x \geq b^0 \\ & \forall (A,b) \in S \; \exists y \in \mathbb{R}^{\hat{n}} : \\ & \qquad Ax + \hat{A}y \geq b \\ & \qquad d'y \leq D \end{aligned}$$

When S is a closed set in the vector space $\mathbb{R}^{(m \times n+m)}$ we know that either the infimum is attained, or the problem is unbounded. This case constitutes the principal object of our considerations, the Linear Recovery Robust Program:

Definition 5. *Let A^0 be an $m \times n$-matrix called the* nominal matrix*, b^0 be an m-dimensional vector called the* nominal right-hand side*, c be an n-dimensional vector called the* nominal cost vector*, $\hat{A}$ be an $m \times \hat{n}$-matrix called the* recovery matrix*, d be an $\hat{n}$-dimensional vector called the* recovery cost vector*, and D be a non-negative number called the* recovery budget*. Further let S be a closed set of pairs of $m \times n$-matrices and m-dimensional vectors, called the* scenario set*. Then the following optimization problem is called a* Linear Recovery Robust Program (LRP) *over S:*

$$\begin{aligned}
&\min_x c'x \\
&\quad s.t. \qquad A^0x \geq b^0 \\
&\qquad \forall (A,b) \in S \; \exists y \in \mathbb{R}^{\hat{n}} : \\
&\qquad\qquad\qquad\qquad Ax + \hat{A}y \geq b \\
&\qquad\qquad\qquad\qquad d'y \leq D
\end{aligned}$$

We refer to the A as the *planning matrix* although it is a quantified variable. The planning matrix describes how the planning x influences the feasibility in the scenario. The vectors $y \in \mathbb{R}^{\hat{n}}$ with $d'y \leq D$ are called the *admissible recovery vectors*. Note that we do not call S a scenario *space*, because primarily there is no probability distribution given for it.

We are not unnecessarily restrictive, when requiring the same number of rows for A^0 as for $\hat{A}$ and A. If this is not the case, nothing in what follows is affected, except may be readability.

If a solution x can be recovered by an admissible recovery y in a certain scenario s, we say x *covers* s.

To any LRP we can associate a linear program, which we call the scenario expansion of the LRP:

$$\begin{aligned}
\min_{x,(y^s)_{s\in S}} \quad & c'x \\
\text{s.t.} \quad & A^0x \geq b^0 \\
& A^sx + \hat{A}y^s \geq b^s \;\; \forall s \in S \\
& d'y^s \leq D \qquad \forall s \in S
\end{aligned}$$

Note that in this formulation the set S is comprised of the scenarios s, whereas in the original formulation it contains (A^s, b^s). This ambiguity of S is convenient and should cause no confusion to the reader. Further, note that in the scenario expansion of an LRP each recovery variable y^s is indexed by its scenario. Thus the solution vector to the scenario expansion contains for each scenario a separate copy of the recovery vector. In the original formulation the recovery vector y is not indexed with a scenario, because the formulation is not a linear program but a logical expression where y is an existence quantified variable.

The scenario expansion is a first possibility to solve the LRP. But, usually, the scenario set is too big to yield a solvable scenario expansion. The scenario sets, which we will consider, are not even finite.

We will frequently use an intuitive reformulation of an LRP, that can be interpreted as a game of a *planning player* setting x, a *scenario player* choosing (A, b), and a *recovery player* deciding on the variable y. The players act one after the other:

$$\inf_x c'x \text{ s.t. } A^0x \geq b^0 \wedge D \geq \left\{ \sup_{(A,b)\in S} \left\{ \inf_y d'y \text{ s.t. } Ax + \hat{A}y \geq b \right\} \right\} \tag{20}$$

with constant vectors $c \in \mathbb{R}^n$, $b^0 \in \mathbb{R}^m$ and $d \in \mathbb{R}^{\hat{n}}$, constant matrices $A^0 \in \mathbb{R}^{m,n}$ and $\hat{A} \in \mathbb{R}^{m,\hat{n}}$, and variables $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m,n}$, $b \in \mathbb{R}^m$ and $y \in \mathbb{R}^{\hat{n}}$.

Again, when it is clear that either the extrema exist or the problem is unbounded we use the following notation:

$$\min_x c'x \text{ s.t. } A^0x \geq b^0 \wedge D \geq \left\{ \max_{(A,b)\in S} \left\{ \min_y d'y \text{ s.t. } Ax + \hat{A}y \geq b \right\} \right\} \tag{21}$$

Observe, that an LRP, its scenario expansion and its 3-player formulation have the same feasible set of planning solutions x. Whereas, the set of recovery vectors y, that may occur as a response to some scenario (A^s, b^s) in the 3-player formulation, is only a subset of the set of feasible second stage solutions y^s in the scenario expansion. The 3-player formulation restricts the later set to those responses y, which are minimal in $d'y$. But this does not affect the feasible set for x.

The formalism of Problem (21) can also be used to express, that x and y are required to be non-negative. But it is a lot more well arranged, if we state such conditions separately:

$$\min_x c'x \text{ s.t. } A^0x \geq b^0 \wedge D \geq \left\{ \max_{(A,b)\in S} \left\{ \min_{y\geq 0} d'y \text{ s.t. } Ax + \hat{A}y \geq b \right\} \right\} \tag{22}$$

and

$$\min_{x\geq 0} c'x \text{ s.t. } A^0x \geq b^0 \wedge D \geq \left\{ \max_{(A,b)\in S} \left\{ \min_{y\geq 0} d'y \text{ s.t. } Ax + \hat{A}y \geq b \right\} \right\} \tag{23}$$

The purely deterministic condition $A^0x \geq b^0$, which we call *nominal feasibility* condition, could also be expressed implicitly by means of S and $\hat{A}$. But, this would severely obstruct readability. In some applications the nominal feasibility plays an important role. For example, a delay resistant timetable shall be feasible for the nominal data, i.e., it must be possible to operate the published timetable unchanged at least under standard conditions. Else, trains could be scheduled in the published timetable x to depart earlier from a station than they arrive there. However, in this rather technical section the nominal feasibility plays a minor role.

Let us mention some extensions of the model. The original problem may as well be an integer or mixed integer linear program,

$$\min_{x=(\tilde{x},\bar{x}),\bar{x}\in\mathbb{Z}} c'x \text{ s.t. } A^0x \geq b^0 \wedge D \geq \left\{ \max_{(A,b)\in S} \left\{ \min_y d'y \text{ s.t. } Ax + \hat{A}y \geq b \right\} \right\} \tag{24}$$

or some other optimization problem over a set of feasible solutions P and an objective function $c : \mathbb{R}^n \to \mathbb{R}$, in case the *disturbances* are confined to the right-hand side:

$$\inf_{f\in P} c(f) \text{ s.t. } D \geq \left\{ \sup_{b\in S} \left\{ \inf_y d'y \text{ s.t. } f + \hat{A}y \geq b \right\} \right\} \tag{25}$$

with a fixed planning matrix A.

Using the concept of planned limits to the recovery cost (cf. p. 6), the budget D can also play the role of a variable:

$$\min_{D\geq 0,\, x} c'x + D \text{ s.t. } A^0x \geq b^0 \wedge$$
$$D \geq \left\{\max_{(A,b)\in S} \left\{\min_y d'y \text{ s.t. } Ax + \hat{A}y \geq b\right\}\right\} \tag{26}$$

In case of right-hand side disturbances only, we can again formulate:

$$\inf_{f\in P, D\geq 0} c(f) + D \text{ s.t. } D \geq \left\{\sup_{b\in S} \left\{\inf_y d'y \text{ s.t. } f + \hat{A}y \geq b\right\}\right\} \tag{27}$$

4.2 Solving Right-Hand Side LRPs

In this part we show that some scenario sets for the right-hand side data of an LRP yield problems that can be solved by a relatively small linear program.

Consider again the 3-player formulation of an LRP (21). Let $P := \{x \in \mathbb{R}^n : A^0x \leq b^0\}$ be the polytope of nominally feasible solutions. If we fix the strategies of the first two players, i.e., the variables x and (A, b), we get the recovery problem of the LRP: $\min d'y$ subject to $\hat{A}y \geq b - Ax$. The dual of the latter is $\max_{\zeta\geq 0}(b - Ax)'\zeta$ s.t. $\hat{A}'\zeta \leq d$. The recovery problem is a linear program. Thus, we have strong duality, and replacing this linear program by its dual in expression (21) will not change the problem for the players optimizing x respectively (A, b).

$$\min_{x\in P} c'x \text{ s.t. } D \geq \left\{\max_{(A,b)\in S} \left\{\max_{\zeta\geq 0}(b - Ax)'\zeta \text{ s.t. } \hat{A}'\zeta \leq d\right\}\right\} \Leftrightarrow$$
$$\min_{x\in P} c'x \text{ s.t. } D \geq \left\{\max_{(A,b)\in S,\zeta\geq 0}(b - Ax)'\zeta \text{ s.t. } \hat{A}'\zeta \leq d\right\} \tag{28}$$

Consider the maximization problem in formulation (28) for a fixed x, thus find $\max_{(A,b)\in S,\zeta\geq 0}(b-Ax)'\zeta$ subject to $\hat{A}'\zeta \leq d$. Assume for a moment $\|b-Ax\|_1 \leq \Delta$. In this case, for each fixed vector ζ the maximum will be attained, if we can set $\text{sign}(\zeta_i)(b-Ax)_i = \Delta$ for i with $|\zeta_i| = \|\zeta\|_\infty$ and 0 else. In other words, under the previous assumptions $(b - Ax)'\zeta$ attains its maximum when $(b - Ax) = \Delta e_i$ for some suitable $i \in [m]$. Therefore, if we have $\|b - Ax\|_1 \leq \Delta$, we can reformulate problem (28):

$$\min_{x\in P} c'x \text{ s.t. } \forall i \in [m] : D \geq \left\{\max_{\zeta\geq 0}(\Delta e_i)'\zeta \text{ s.t. } \hat{A}'\zeta \leq d\right\} \Leftrightarrow$$
$$\min_{x\in P} c'x \text{ s.t. } \forall i \in [m] : D \geq \left\{\min_y d'y \text{ s.t. } \hat{A}y \geq \Delta e_i\right\} \tag{29}$$

The at first sight awkward condition $\|b - Ax\|_1 \leq \Delta$ is naturally met if only the right-hand side data changes, and is limited in the set $S^1 := \{(A^s, b^s) :$

$\|b^* - b^s\|_1 \leq \Delta, A^* = A\}\}$. For an LRP over S^1 formulation (29) is equivalent to a linear program of size $\mathcal{O}(m(n + \hat{n} \cdot m))$:

$$\begin{aligned} \min_{x \in P} \quad & c'x \\ \text{s.t.} \quad & \forall i \in [m]: \\ & Ax + \hat{A}y^i \geq b + \Delta e_i \\ & d'y^i - D \geq 0 \end{aligned}$$

Next, consider the scenario set

$$S^k := \{(A^s, b^s) : \|b^* - b^s\|_1 \leq k \cdot \Delta, \|b^* - b^s\|_\infty \leq \Delta, A^* = A\}$$

for arbitrary $k > 1$. By the same token, the maximization over ζ in formulation (28) for fixed x and (A, b) can be achieved, by setting the maximal $\lfloor k \rfloor$ entries of the vector $(b - Ax)$ equal to 1 and the $\lceil k \rceil$-th entry equal to $k - \lfloor k \rfloor$. For example, when k is integer, we can replace the scenario set S^k by those $\binom{k}{m}$ scenarios, where exactly k entries of b deviate maximally from b^*, and the other entries equal their reference value b_i^*. So, we have:

Theorem 1. *An LRP over S^k can be solved by a linear program of size polynomial in n, $\hat{n}$, m, and $\binom{k}{m}$.*

Corollary 1. *An LRP over S^1 can be solved by a linear program of size polynomial in n, $\hat{n}$, and m.*

Corollary 2. *For fixed k an LRP over S^k can be solved by a linear program of size polynomial in n, $\hat{n}$, and m.*

Of course, in practice this approach will only work, when k is very small.

The above reasoning can give a fruitful hint to approach RROPs in general. First, try to find a small subset of the scenario set, which contains the worst-case scenarios, and then optimize over this set instead of the whole scenario set. In the above setting we can achieve this very easily, because the recovery problem fulfills strong duality. If the recovery problem is an integer program this approach fails in general. Still, one can try to find a small set of potential worst case scenarios, to replace the original scenario set. Unlike the recovery problem, the planning problem may well be an integer or mixed integer program, as we show in the following.

For the manipulations of the formulations the linearity of $P, Ax \geq b$ or c is immaterial. So we can extend the above reasoning to non-linear optimization problems. Let $c : \mathbb{R}^n \to \mathbb{R}$ be a real function, P' a set of feasible solutions and $\{g_i : \mathbb{R}^n \to \mathbb{R}\}_{i \in [m]}$ be a family of real functions, and assume that extrema in the resulting RROP are either attained, or the problem is unbounded. For the scenario set S^1 of right-hand side disturbances we have with the above notation

$$\begin{gathered}\min_{x\in P'} c(x)\\ \text{s.t. } D \geq \left\{\max_{b\in S^1} \left\{\min_y d'y \text{ s.t. } g(x) + \hat{A}y \geq b\right\}\right\}\\ \Leftrightarrow\\ \min_{x\in P'} c(x)\\ \text{s.t. } \forall i \in [m] : g(x) + \hat{A}y^i \geq b^* + \Delta e_i\\ D - d'y^i \geq 0\end{gathered}$$

In particular we are interested in the case of an integer linear program ($\min c'x$, $Ax \geq b, x \in \mathbb{Z}^n$) with right-hand side uncertainty. We get as its recovery robust version over S^1.

$$\begin{gathered}\min_{x\in\mathbb{Z}^n} c'x\\ \text{s.t. } A^0x \geq b^0\\ D \geq \left\{\max_{b\in S^1} \left\{\min_y d'y \text{ s.t. } A^*x + \hat{A}y \geq b\right\}\right\}\\ \Leftrightarrow\\ \min_{x\in\mathbb{Z}} c'x\\ \text{s.t. } A^0x \geq b^0\\ \forall i \in [m] : A^*x + \hat{A}y^i \geq b^* + \Delta e_i\\ D - d'y^i \geq 0\end{gathered}$$

Let $A^* = A^0$ and $b^* = b^0$. Defining $f := A^*x - b^*$ we can rewrite the previous program as

$$\begin{gathered}\min_{(x,f)\in\mathbb{Z}^{n+m}} \tilde{c}'(x,f)\\ A^*x - f = b^*\\ \text{s.t. } \forall i \in [m] : f + \hat{A}y^i \geq \Delta e_i\\ d'y^i - D \geq 0\\ f \geq 0\end{gathered} \tag{30}$$

With a suitable cost vector $\tilde{c}$. Note that the original integer linear program corresponds with the scenario part of the program only via the slack variable f. In other words, for solving the recovery robust version the solving procedures for the original, deterministic, integer linear optimization problem can be left untouched. We only have to flange a set of linear inequalities to it. The f variables function as means of communication between the original integer problem, where they correspond to the slack in each row, and the linear part, in which their effect on robustness is evaluated. In the next part we will consider this communication situation for an even more specialized type of recovery.

4.3 Robust Network Buffering

Let us use Corollary 1 for the Simple Robust Timetabling problem with right-hand side uncertainty limited in S^1. Set $g_2 = 0$ to drop the limit to the maximal

delay at a node. By the corollary the Simple Robust Timetabling problem over S^1 reads as follows. Let χ_a be the indicator function of a, i.e., $\chi_a(x) = 1$ if $a = x$, and zero else.

$$\begin{aligned} &\min_{\pi,f} \textstyle\sum_{e=(i,j)\in E} w(e)(\pi_j - \pi_i) \\ &\text{s.t.} \qquad \pi_j - \pi_i + f_e = t(e), \qquad \forall e = (i,j) \in E \\ &\forall s \in E: \\ &\qquad f_e + y_j^s - y_i^s \geq \Delta \cdot \chi_e(s),\ \forall e = (i,j) \in E \\ &\qquad D - d' y^s \geq 0 \\ &\qquad f, y^s \geq 0 \end{aligned} \tag{31}$$

Periodic Timetabling. Many service providers operate periodic schedules. This means that—during some period of the day—equivalent events, e.g., all departures of the trains of a certain line at a certain station take place in a periodic or almost periodic manner. For example, at a subway station each departure will take place exactly, e.g., 10 minutes after the departure of the previous train. Likewise, most timetables for long-distances connections are constructed such that if a train leaves from the central station of X to central station of Y at 12:43h, then the next train to Y will leave the central station of X at (roughly) 13:43h, the next at 14:34h, and so on. This means that the long-distance connection from X to Y is operated with a *period* of one hour.

In case of periodic timetables, we do not plan the single events as in the aperiodic case, but we plan periodic events. For these we schedule a periodic time, which is understood modulo the period of the system. (There may also be differnt periods in the same system, but we restrict our consideration here to the case of a single, global period.) Assume we assign the value 5 to the variable corresponding to the periodic event that trains of line A depart from station S towards station S'. Let the period T of the system be one hour. Then—in every hour—five minutes past the hour a train of line A will depart from station S towards station S'. This leads to the Periodic Event Scheduling Problem (PESP)[2], which can be formulated as a mixed integer program of the following form. Let $G(A, V)$ be a directed graph and three functions $w, u, l : A \to \mathbb{R}$ on the arc set. Then the following problem is called a PESP.

$$\begin{aligned} &\min_{k\in\mathbb{Z}^{|A|},\pi} \textstyle\sum_{e=(i,j)\in A} w(e)(\pi_j - \pi_i + k_e T) \\ &\text{s.t.} \quad u(e) \geq \pi_j - \pi_i + k_e T \geq l(e), \quad \forall e = (i,j) \in A \end{aligned}$$

This type of problem has a broad modeling power. For a comprehensive study on periodic timetabling we refer the interested reader to [10].

To construct an RROP from an original problem, which is a PESP we have to make a choice, whether we interpret the disturbances as periodic disturbances,

[2] The Periodic Event Scheduling Problem was introduced in [14]. For details confer also [10].

like a construction site, that will slow down the traffic at a certain point for the whole day, or as aperiodic events, like a jammed door at a stopping event. For periodic disturbances we get the following program.

$$\begin{aligned}
\min_{k\in\mathbb{Z}^{|A|},\pi,f} \quad & \textstyle\sum_{e=(i,j)\in A} w(e)(\pi_j - \pi_i + k_e T) \\
\text{s.t.} \quad & \pi_j - \pi_i + f_e + k_e T = l(e), && \forall e=(i,j)\in A \\
& \pi_j - \pi_i + \bar{f}_e + k_e T = u(e), && \forall e=(i,j)\in A \\
\forall s \in A, \Xi \in \{0,1\}: & \\
& f_e + y^s_j - y^s_i \geq \Delta \cdot \chi_e(i) \cdot \Xi, && \forall e=(i,j)\in A \\
& \bar{f}_e + y^s_i - y^s_j \geq \Delta \cdot \chi_e(i) \cdot (1-\Xi), && \forall e=(i,j)\in A \\
& D - d' y^s \geq 0 \\
& y^s \geq 0
\end{aligned}$$

Note that the right-hand sides are still constants, though they look like a quadratic term.

Again, the deterministic PESP instance can be flanged with a polynomial size linear program to ensure robustness. This structure can be helpful for solving such a problem, as the specialized solving techniques for the original integer program can be integrated.

As an example for this approach confer [3], where a specialized technique for an advanced platforming problem was combined with robust network buffering to get a recovery robust platforming. The method was tested on real-world data of Italian railway stations. The propagated delay through the stations was reduced by high double-digit percentages without loss in the primal objective, which is to maximizes the number of trains the station handles.

General Network Buffering. The general situation is the following: We are given an optimization problem on a network. The solution to that problem will be operated under disturbances. The disturbances propagate through the network in a way depending on the solution of the optimization problem. The solution of the original optimization problem x translates into a buffer vector f on the arcs of the network. Changing perspective, the original problem with its variables x is a cost oracle: If we fix a certain buffering f, the optimization will construct the cheapest x vector to ensure the buffering f. Let us summarize the general scheme.

Given an optimization problem P with the following features:

- A directed graph G.
- An unknown, limited, nonnegative vector of disturbances on the arcs, or on the nodes, or both.
- The disturbances cause costs on the arcs, or on the nodes, or both, which propagate through the network.
- A vector of absorbing potential on the arcs, the nodes, or both can be attributed to each solution of P.

If we further restrict the disturbance vector to lie in S^1, we get the following by the above considerations: The recovery robust version of P, in which the propagated cost must be kept below a fixed budget D, can be formulated as the original problem P plus a linear program quadratic in the size of G.

5 Platforming: A Real-World Study

In the previous section we have shown that a method to solve a linear, or convex, or linear integer optimization problem can be extended to a method for the recovery robust version of the problem with right-hand side disturbances in an efficient and simple way, provided that the recovery can be described by a linear program. Those conditions are fulfilled in particular for network buffering problems. In this case the recovery is the propagation of a disturbance along a network. The propagation in the network depends on the solution of the original optimization problem. But this original optimization problem itself need *not* be a linear program.

The advantage of this method for robustness is threefold:

- It yields an efficient algorithm, respectively it does not add to the complexity of the original problem.
- The method provides for solutions which possess a precisely defined level of robustness (in contrast to heuristics).
- The method is easy to implement. One can reuse any existing approach for the original problem and supplement it with a linear program for recoverable robustness.

To exemplify these advantages we describe a study on real-world data for the train platforming problem (cf. [3]).

The train platforming problem considers a single station and a given set of trains together with their planned departure and arrival times at the station area. The goal is a conflict-free assignment of a *pattern* to as many of these trains as possible. A pattern consists of a track in the station together with an arrival path to this track and a departure path from the track. The assignment is conflict free, if no track has a time interval during which two trains are assigned to that track, and no pair of simultaneously used paths are in spacial conflict. In the current study the spacial conflicts of paths are given explicitly in a conflict graph.

It is straight forward to formulate this problem as an integer linear program with $(0, 1)$-decision variables for each pair of a pattern and a train. Of course, there are several possibilities to phrase this problem as an integer linear program. In fact, the version used in the study is not trivial, but constructed carefully to achieve a powerful model that allows to solve large-scale instances. (For details we refer the reader to [3].) Independent of this particular study one might use a different integer programming formulation, e.g., in case the path conflicts are not given as a conflict graph, but implicitly by a digraph representing the infrastructure network. But the particular program is not relevant to the general approach on which we focus here.

The original program is reused without changes in the recovery robust program. To effect robustness we add a linear program modeling the delay propagation and a set of constraints that link the variables of the delay propagation to those of the original program. The resulting program has three sections:

1. The original train platforming program to optimize the assignment of patterns to trains (planning sub-model).
2. The linear program to model the delay propagation network (recovery sub-model).
3. The constraints linking the nominal solution to the buffer values on the delay propagation network (linking constraints).

In the delay propagation network each train has three vertices corresponding to the three events in which it will free up each of the three resources assigned to it (arrival path, platform, and departure path). Naturally, two vertices are connected by an arc whenever delay at the train-resource pair corresponding to the head-node may propagate onto the tail-node for a specic nominal solution. A delay in freeing up the platform for a train may propagate to a delay in freeing up the same platform for other trains. Something similar applies to arrival/departure paths, more precisely for paths that are in conflict. Every arc in the delay propagation network has an associated buffer value, which represents the maximum amount of delay that it is able to absorb without any propagation effect. Intuitively, a buffer corresponds to the slack among a given pair of resource occupation time intervals.

The objective function of the original problem contains three parts that are weighted (in the order given below) such that the optimal solution will also be lexicographically optimal.

1. The total number of trains that can be assigned.
2. Certain trains have a preferred set of tracks to one of which they should be assigned if possible.
3. A heuristic for robustness punishing any use of pairs of paths that are in spacial conflict during time windows that are not overlapping (i.e. the assignment is conflict-free) but close to each other.

For the recovery robust version we drop the third, heuristic objective and replace it by the exact objective to minimize the maximum delay that can occur. We use the scenario set S^1 to get a compact model for the robust platforming. It turns out that the second objective plays a role for none of the two methods in any of the considered instances, i.e., the trains that are assigned to tracks can always be assigned to their preferred tracks. Moreover, the real-world instances are such that it is not possible to assign tracks to all given trains, neither in the standard nor in the recovery robust model.

Note that by the weighting of the objective function this implies, that a conflict-free assignment of all trains is physically impossible. But, in all considered instances the recovery robust method assigns as many trains as the original method, i.e., as much as possible in general. Thus the two methods yield assignments that are equivalent in all given deterministic criteria. But they differ

Table 1. Results for Palermo Centrale

time window	# trains not platformed	D nom	CPU time nom (sec)	D RR	CPU time RR (sec)	Diff. D	Diff. D in %
A: 00:00-07:30	0	646	7	479	46	167	25.85
B: 07:30-09:00	2	729	7	579	3826	150	20.58
C: 09:00-11:00	0	487	6	356	143	131	26.90
D: 11:00-13:30	2	591	6	384	228	207	35.03
E: 13:30-15:30	1	710	9	516	2217	194	27.32
F: 15:30-18:00	1	560	7	480	18	80	14.29
G: 18:00-00:00	3	465	11	378	64	87	18.71

significantly in delay propagation. In all instances the recovery robust method yields assignments with a double-digit percentage of delay reduction. In one case the reduction is almost 50%. Averaged over all instances the reduction is roughly 1/4.

Table 1 gives the details of the study for the station Palermo Centrale. The study considers seven time windows during the day at the station Palermo Centrale. These are given in the first column of Table 1. Further, the short cut *nom* denotes values referring to the original method, whereas *RR* stands for the results of the recovery robust approach. For both the table states the CPU time required to find the solution and the maximal propagated delay. Further, we give the difference in propagated delay as absolute value (in minutes) and as percentage of delay propagation in the original method's solution. The number of non-assigned trains is given without reference to the method, because both methods achieve the same value here.

6 Conclusion

We have introduced *recoverable robustness* as an alternative concept for optimization under imperfect information. It is motivated by practical problems like delay resistant timetabling, for which classical concepts like stochastic programming and robust optimization prove inappropriate. We describe the model in full generality and demonstrate how different types of delay resistant timetabling problems can be modeled in terms of recoverable robustness. Further, we specialized the general concept of recoverable robustness to *linear recovery robust programs*. For these we provide an efficient algorithm in case of right-hand side disturbances. By means of this general method delay resistant timetabling problems can be solved efficiently. This is exemplified by a real world application of our method in a study [3] on recovery robust platforming. The platformings constructed with our method achieve maximal possible throughput at the stations, but drastically reduces the delay propagation in comparison to a state-of-the-art method.

References

1. Special issue on robust optimization. Mathematical Programming A 107(1-2) (2006)
2. Berger, A., Lorenz, U., Hoffmann, R., Stiller, S.: TOPSU-RDM: A simulation platform for railway delay management. In: Proceedings of SimuTools (2008)
3. Caprara, A., Galli, L., Stiller, S., Toth, P.: Recoverable-robust platforming by network buffering. Technical Report ARRIVAL-TR-0157, ARRIVAL Project (2008)
4. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: On the interaction between robust timetable planning and delay management. Technical Report ARRIVAL-TR-0116, ARRIVAL project (2007); Published at COCOA 2008
5. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Robust algorithms and price of robustness in shunting problems. In: Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, ATOMS (2007)
6. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Recoverable robust timetabling: Complexity results and algorithms. Technical report, ARRIVAL Project (2008)
7. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A., Schachtebeck, M., Schöbel, A.: Recoverable robustness in shunting and timetabling. In: Robust and Online Large-Scale Optimization, pp. 29–61. Springer, Heidelberg (2009)
8. Cicerone, S., Di Stefano, G., Schachtebeck, M., Schöbel, A.: Dynamic algorithms for recoverable robustness problems. In: Proceedings of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2008), Schloss Dagstuhl Seminar Proceedings (2008)
9. Cicerone, S., Di Stefano, G., Schachtebeck, M., Schöbel, A.: Multi-stage recoverable robustness problems. Technical report, ARRIVAL Project (2009)
10. Liebchen, C.: Periodic Timetable Optimization in Public Transport. dissertation.de – Verlag im Internet, Berlin (2006)
11. Liebchen, C., Lübbecke, M., Möhring, R.H., Stiller, S.: Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL-Project (2007)
12. Liebchen, C., Stiller, S.: Delay resistant timetabling. Preprint 024–2006, TU Berlin, Institut für Mathematik (2006)
13. Tyrrell Rockafellar, R.: Network ows and monotropic optimization. John Wiley & Sons, Inc., Chichester (1984)
14. Serafini, P., Ukovich, W.: A mathematical model for periodic scheduling problems. SIAM Journal on Discrete Mathematics 2(4), 550–581 (1989)
15. Soyster, A.L.: Convex programming with set-inclusive constraints and applications to inexact linear programming. Operations Research 21(4), 1154–1157 (1973)
16. Stiller, S.: Extending Concepts of Reliability. Network Creation Games, Real-time Scheduling, and Robust Optimization. TU Berlin, Dissertation, Berlin (2009)
17. Vromans, M., Dekker, R., Kroon, L.: Reliability and heterogeneity of railway services. European Journal of Operational Research 172, 647–665 (2006)