Recoverable Robustness for Scheduling with Precedence Constraints

Alberto Caprara[⋆] Laura Galli[⋆] Sebastian Stiller[⋄] Paolo Toth[⋆]

[⋆] *DEIS, University of Bologna*
Viale Risorgimento, 2 40136 Bologna, Italy

[⋄] *Institut für Mathematik, Technische Universität Berlin*
Straße des 17. Juni, 136 10623 Berlin, Germany

Abstract

In this paper we apply the concept of *recoverable robustness by network buffering* in a real-world context for the first time. In particular, we consider the problem of assigning trains to platforms in a large railway station, and extend the mathematical formulation of the "nominal" problem with variables and constraints that allow us to limit the propagation of delays that may occur at some point in the schedule. Our experimental results show that the resulting solutions have the same quality in terms of the nominal objective function, whereas delay propagation is significantly reduced.

Keywords: *Robustness, Network Buffering, Train Platforming*

1 Introduction

Robust optimization is an approach to optimization under uncertainty, aimed at seeking for good solutions although the input data, or part of the input data, is not known exactly. In classical robust optimization, one looks for a solution which is feasible for all possible scenarios. Often such a solution may not exist at all, or it may come at an unacceptably high nominal cost. To overcome this serious weakness of classical robustness, the concept of recoverable robustness has been introduced in [3]. A solution x together with an algorithm A is *recovery robust* for a certain set of scenarios $\mathcal{S}$ and a certain family of admissible recovery algorithms $\mathcal{A}$, if $A \in \mathcal{A}$ and for each scenario in $\mathcal{S}$, application of algorithm A to the solution x yields a solution x' which is feasible for the scenario. Thus, recoverable robustness allows for *limited* second-stage decisions, the so-called recovery.

An important special case mentioned in [3] occurs when the nominal problem input contains the time instants at which a given set of events happen, the scenarios are associated with disturbances over these time intervals, and the (unique) recovery algorithm adjusts a nominal solution x by propagating the delays over a suitable acyclic network in which precedence constraints are imposed by x, shifting forward the events in time, so as to get the desired x'. In this case, the algorithm calls for the solution of a linear program (depending on x) and the set of possible scenarios correspond to variations of the associated right-hand-side, with an upper bound on the overall sum of these variations. This general method, called *network buffering*, has been proposed in [3] and to the best of our knowledge never tested experimentally on a real-world case study. Classical robust approaches [1] cannot model this case since there are uncountably many scenarios.

An example of a real world problem with the above type of scenarios and recovery, and the need for robustness without over-conservative solutions, is the Train Platforming Problem (TPP). The TPP consists in assigning trains to platforms in a railway station and defining the corresponding routes (arrival and departure paths) in the station area [2]. For TPP the recovery amounts to *propagating* delay through a certain network according to a buffering which results from the platforming.

Moving from theory to practice, our aim in this paper is to see whether network buffering is doable in a real-life context and experiment its effectiveness. Experiments prove that the model can be solved for real-world instances. The robust solutions produce *significantly* less delay propagation, than those optimized for the nominal objective, while *maintaining* nominal optimality. In other words the *price of robustness* [3] is 1, while delay reduces by a double-digit percentage.

2 Recoverable robustness and network buffering

Robust optimization is an approach to optimization under uncertainty, aimed at seeking for good solutions although the input data, or part of the input data, is not known exactly. In classical robust optimization, one looks for a solution which is feasible for all possible scenarios. Often such a solution may not exist at all, or it may come at an unacceptably high nominal cost. To overcome this serious weakness of classical robustness, the concept of recoverable robustness has been introduced in [3]. A solution x together with an algorithm A is *recovery robust* for a certain set of scenarios $\mathcal{S}$ and a certain family of admissible recovery algorithms $\mathcal{A}$, if $A \in \mathcal{A}$ and for each scenario in $\mathcal{S}$, application of algorithm A to the solution x yields a solution x' which is feasible for the scenario. Thus, recoverable robustness allows for *limited* second-stage decisions, the so-called recovery.

An important special case mentioned in [3] occurs when the nominal problem input contains the time instants at which a given set of events happen, the scenarios are associated with disturbances over these time intervals, and the (unique) recovery algorithm adjusts a nominal solution x by propagating the delays over a suitable acyclic network in which precedence constraints are imposed by x, shifting forward the events in time, so as to get the desired x'. In this case, the algorithm calls for the solution of a linear program (depending on x) and the set of possible scenarios correspond to variations of the associated right-hand-side, with an upper bound on the overall sum of these variations. This general method, called *network buffering*, has been proposed in [3] and to the best of our knowledge never tested experimentally on a real-world case study. Classical robust approaches [1] cannot model this case since there are uncountably many scenarios.

An example of a real world problem with the above type of scenarios and recovery, and the need for robustness without over-conservative solutions, is the Train Platforming Problem (TPP). The TPP consists in assigning trains to platforms in a railway station and defining the corresponding routes (arrival and departure paths) in the station area [2]. For TPP the recovery amounts to *propagating* delay through a certain network according to a buffering which results from the platforming.

Moving from theory to practice, our aim in this paper is to see whether network buffering is doable in a real-life context and experiment its effectiveness. Experiments prove that the model can be solved for real-world instances. The robust solutions produce *significantly* less delay propagation, than those optimized for the nominal objective, while *maintaining* nominal optimality. In other words the *price of robustness* [3] is 1, while delay reduces by a double-digit percentage.

3 Recoverable robustness and network buffering

In this paper, we restrict our attention to the following case of recoverable robustness.

First, the input of the nominal problem includes a set of *events* that take place at some point in time. For instance, in TPP these events correspond to a train arriving (at the station) or a train departing (from the station), according to a predefined schedule.

Second, the scenarios are determined by "external" disturbances that cause delay in these events, and there is an upper bound on the total amount of external disturbances. For instance, a train may arrive with 5 minutes delay since it departed late from its previous station, or it may arrive on time but depart with 5 minutes delay due to problems in changing the crew.

Third, these delays propagate along an acyclic delay propagation network, in which nodes correspond to events and, for each arc (e_1, e_2), the delay that propagates from event e_1 to event e_2 depends on the specific solution of the nominal problem. For instance, if e_1 corresponds to the departure of train 1, equal to 10:00 in the nominal problem input, and e_2 corresponds to the arrival of train 2, equal to 10:05 in the nominal problem input, in case the nominal solution assigns these two trains to the same platform, a delay of 8 minutes for event e_1 propagates to a delay of 3 minutes for event e_2. We also say that the arc (e_1, e_2) can *absorb* 5 minutes of delay in e_1 when this is propagated to e_2. On the other hand, if the two trains are assigned to different platforms (and to compatible arrival/departure paths, as explained in the following) there is no delay that propagates from e_1 to e_2.

Another interpretation of the delay propagation mechanism, which allows us to fit our case into the general framework of recoverable robustness, is that there is a unique recovery algorithm $A \in \mathcal{A}$, which simply takes a solution of the nominal problem (feasible in absence of delays) and, taking into account the external delays occurring in the specific scenario, shifts forward in time each event "as little as possible" so as to get a feasible solution. Note that this is always possible if the time horizon is linear, i.e., we are not considering a cyclic schedule. Nothing else in the solution is changed; for instance the assignment of trains to platforms is not modified by A.

3.1 Formal definition of $\mathcal{S}$ and A

Formally, let E be the set of events in the problem and $N = (E, A(E))$ be the delay propagation network. External disturbances are modeled via values $\delta_e \geq 0$ assigned to each event $e \in E$, and the condition that the total amount of external disturbances is bounded is formalized by $\sum_{e \in E} \delta_e \leq \Delta$. In other words, the set of feasible scenarios $\mathcal{S}$ is given by:

$$\{\delta \in \mathbb{R}^E_+ : \sum_{e \in E} \delta_e \leq \Delta\} \tag{1}$$

Delay propagations are represented by the so-called *buffer values* $f_{(e_1,e_2)}$ associated with the arcs in D, representing the delay that can be absorbed when it propagates from event e_1 to event e_2 for a specific nominal solution. In other words, given a nominal solution, algorithm A finds an optimal solution to the following linear program with variables d_e, each representing the delay of event $e \in E$:

$$\min \sum_{e \in E} d_e$$

$$\forall e \in E : d_e \geq \delta_e$$

$$\forall (e_1, e_2) \in A(E) : d_{e_2} \geq d_{e_1} - f_{(e_1,e_2)}$$

This problem can be solved by flow techniques; however our main goal here is to "paste" this linear program into the mathematical formulation of the nominal problem, as explained next.

3.2 The recovery-robust overall problem

The simple structure of the recovery algorithm allows us to formulate a recovery-robust version of our nominal problem. Suppose the latter can be formulated as a mathematical problem of the form $\min\{c(x) : x \in F\}$, where $c(\cdot)$ denotes the objective function and F the set of feasible solutions.

In principle, we consider the (uncountably many) feasible scenarios $\xi \in \mathcal{S}$, each associated with external disturbances $\delta^\xi \in \mathbb{R}^E_+$, and introduce the associated delay variables d^ξ_e. Moreover, we introduce the variable D to model the maximum overall delay that is propagated over all scenarios. We add D to the objective function $\min c(x)$, assuming the latter is appropriately scaled to take into account the relative cost of D. The overall model reads:

$$\min c(x) + D$$

$$x \in F$$

$$\forall \xi \in \mathcal{S} : \quad D \geq \sum_{e \in E} d^\xi_e$$

$$\forall \xi \in \mathcal{S},\ \forall e \in E : \quad d^\xi_e \geq \delta^\xi_e$$

$$\forall \xi \in \mathcal{S},\ \forall (e_1, e_2) \in A(E): \quad d^{\xi}_{e_2} \geq d^{\xi}_{e_1} - f_{(e_1,e_2)}$$

$$\forall (e_1, e_2) \in A(E): \quad f_{(e_1,e_2)} = b(x)$$

where the last constraints express the buffer values associated with a specific solution $x \in F$. In other words, the solution of the nominal problem x translates into a buffer vector f on the arcs of the network. The delay propagates in the network until it is absorbed by the buffers.

Of course, the above formulation cannot be approached directly since there are uncountably many scenarios. On the other hand, by a fundamental result of [3] (stated there in a much more general setting), we can restrict attention only to scenarios in which all external disturbances are zero apart from one, which is equal to Δ. In other words, and more formally, we can consider only the vertices of $\mathcal{S}$ as defined in (1) as our scenarios. This leaves only $|E|$ distinct scenarios to consider explicitly in the model above, which is doable in some practical cases.

4 Recovery-robust TPP

We focus our attention on TPP at Rete Ferroviaria Italiana (the Italian infrastructure manager), which consists in assigning trains to platforms in a railway station and defining the corresponding routes (arrival and departure paths), see [2] for a detailed description. The combination of an arrival path, a stopping platform and a departure path for a train is called a *pattern.* The problem is aimed at defining a platforming plan, namely assigning a pattern to each train. The feasibility of the plan is determined by a set of constraints that forbid any overlapping in the occupation of the resources (platforms and routes). These constraints are generally expressed by defining a pattern-incompatibility (undirected) graph, with one node for each pattern and an edge joining each pair of incompatible patterns, as discussed in [2], where the deterministic (or "nominal") version of the problem is considered.

In daily railway operations small, seminal prolongations of driving and stopping activities, so-called disturbances, are inevitable. We aim to limit by means of robust platforming the propagation of these disturbances in the system. To this end, we introduce the following delay propagation network N. Each train has three vertices corresponding to the three events in which it will free up each of the three resources assigned to it (respectively, 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 specific nominal solution, as illustrated above. As already mentioned, 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 the so called incompatible paths, i.e., paths sharing some tracks. Every arc in the 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 windows, e.g., 5 minutes in the example of the previous section.

The final recovery robust optimization model for TPP consists of three main blocks:

1. the original TPP model to optimize the assignment of patterns to trains (*planning* sub-model);
2. 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).

Clearly the propagation in the recovery sub-model, depends on the solution of the planning sub-model. The close interplay between the two sub-models, i.e., how the choice of patterns affects the buffer values, is in this case quadratic. In spite of this, we managed to linearize the linking constraints using the technique described in [2]. What we get is a mixed-integer linear program, which we solve by the branch-and-cut-and-price technique of [2].

4.1 The mathematical program

The objective asks to minimize the cost of the patterns that we choose and the total delay that we propagate over all scenarios. The *recovery robust* version of the TPP model consists of three blocks. The first block represents the nominal problem and contains binary variables x_P that represent the assignment of pattern P to train $t = t(P)$. In this block there are two types of constraints:

- *covering* constraints that require each train to be assigned to a pattern;
- *clique incompatibility* constraints that forbid for each pair of trains to be assigned to incompatible patterns. In order to enforce these constraints instead of expressing the incompatibility for each pair of patterns, we consider the set $\mathcal{K}_{t,t'}$ of all the cliques of incompatible pattern for each pair of distinct trains $(t, t') \in T^2$.

$$\min \sum_{t \in T} \sum_{t(P)=t} c_P x_P + D$$

$$\forall t \in T : \quad \sum_{t(P)=t} x_P = 1$$

$$\forall (t, t') \in T^2, K \in \mathcal{K}_{t,t'} : \quad \sum_{P \in K} x_P \leq 1$$

In the second block we consider the delay propagation network $N = (\{a_t : t \in T\} \cup \{s_t : t \in T\} \cup \{d_t : t \in T\}, A(N))$. There are three types of delays for each train: a_t is the delay on the release time of the arrival path for train t, s_t is the delay on the release time of the platform for train t and finally d_t is the delay on the release time of the departure path for train t. We have five types of arcs in N:

1. arc (a_t, a_t'): represents the propagation of the delay on the release time of the arrival path for train t to the release time of the arrival path for train t'.
2. arc (a_t, s_t'): represents the propagation of the delay on the release time of the arrival path for train t to the release time of the stopping platform for train t'.
3. arc (s_t, a_t'): represents the propagation of the delay on the release time of the stopping platform for train t to the release time of the arrival path for train t'.
4. arc (d_t, a_t'): represents the propagation of the delay on the release time of the departure path for train t to the release time of the arrival path for train t'.
5. arc (d_t, s_t'): represents the propagation of the delay on the release time of the departure path for train t to the release time of the stopping platform for train t'.

The f variables and the D variable have the same meaning as in Section 3.

We consider two types of *external disturbances* respectively $\delta_{t,a}$ on the arrival time of train t and $\delta_{t,s}$ on the departure time of train t. According to the discussion in Section 3, we can restrict attention to the case in which the scenario set $\mathcal{S}$ contains all δ values equal to 0 apart from $\delta_{t,a} = \Delta$ for some $t \in T$, or $\delta_{t,s} = \Delta$ for some $t \in T$. By adding the scenario index ξ for variables a, s, d we get:

$$\forall \xi \in \mathcal{S} : \quad D \geq \sum_{t \in T} (a_t^\xi + s_t^\xi + d_t^\xi)$$

$$\forall \xi \in \mathcal{S}, \ \forall t \in T : \quad a_t^\xi \geq \delta_{t,a}^\xi$$

$$\forall \xi \in \mathcal{S}, \ \forall t \in T : \quad s_t^\xi \geq a_t^\xi + \delta_{t,s}^\xi$$

$$\forall \xi \in \mathcal{S},\ \forall t \in T: \quad d_t^{\xi} \geq s_t^{\xi}$$

$$\forall \xi \in \mathcal{S},\ \forall (h_t, m_{t'}) \in A(N): \quad m_{t'}^{\xi} \geq h_t^{\xi} - f(h_t, m_{t'})$$

Finally in the third block we need the link between the choice of patters, i.e., the solution to the original model, with the corresponding buffer values on the arcs of the delay propagation network. This link is quadratic, namely

$$\forall (h_t, m_{t'}) \in A(N): \quad f(h_t, m_{t'}) = \sum_{t(P)=t} \sum_{t(P')=t'} \alpha_{P,P'} x_P x_{P'}$$

so we use the linearization technique described in [2] which leads to strong bounds.

5 Computational results and conclusions

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

Table 1: Results for Palermo Centrale

In Table 1 we report results for seven time windows of Palermo Central station's timetable. By 'nom' we refer to solutions optimized for the deterministic TPP objective (cf. [2]), whereas 'RR' indicates the recovery robust solutions, for which D is part of the new objective. The parameter D is the maximum propagated delay in minutes over all scenarios with at most 30 minutes of seminal disturbances (i.e., $\Delta = 30$). For each time window the robust solutions manage to assign the same number of trains as the nominally optimal solution. The delay propagation is reduced between 14% up to 35%. Figure 1 is a visualization of the data presented in Table 1.

References

[1] Bertsimas D., Sim M. *The Price of Robustness.* Operations Research, **52(1)** (2004)pp. 35–53.

[2] Caprara A., Galli L., Toth P. *Solution to the Train Platforming Problem* ATMOS 2007.

[3] Liebchen C., Lübbecke M., Möhring R. H., Stiller S. *Recoverable Robustness* ARRIVAL-TR 0066, EU ARRIVAL project.

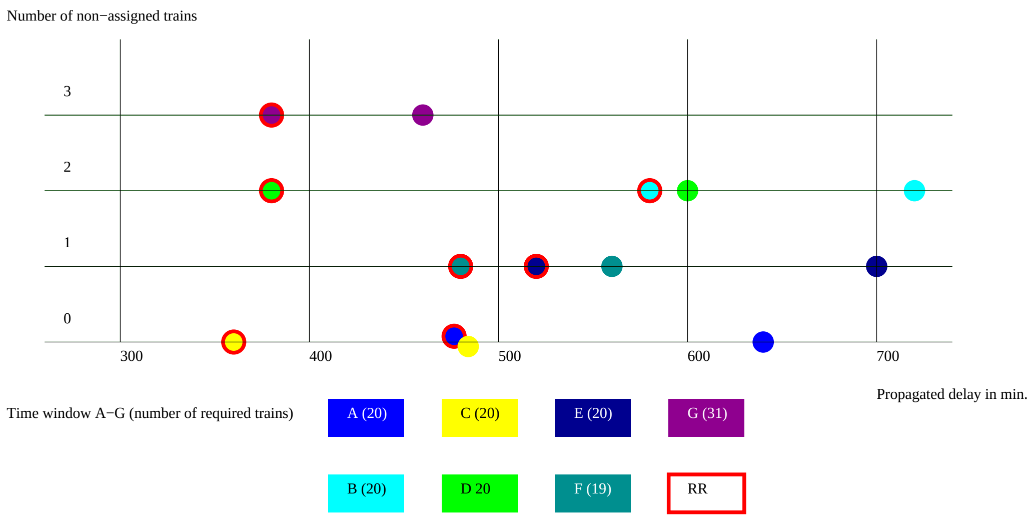

Figure 1: Delay propagation and through-put of robust and non-robust platforming.