Delay-Robust Event Scheduling

Alberto Caprara * Laura Galli * Sebastian Stiller † Paolo Toth *

June 2011

Abstract

Robust optimisation is a well established concept to deal with uncertainty. In particular, *recovery-robust* models are suitable for real-world contexts, where a certain amount of recovery – although limited – is often available. In this paper we describe a general framework to optimise *event-based* problems against *delay propagation*. We also present a real-world application to train platforming in the Italian railways, in order to show the practical effectiveness of our framework.

Keywords: robust optimisation, delay propagation, railway planning

Introduction

For many years, it was assumed that the first thing to consider when tackling an optimisation problem is the complete knowledge of the data. A deterministic model was not regarded as a matter of choice, but as the basis of all possible treatments. A famous remark of Sherlock Holmes reads: "Only one important thing has happened in the last three days, and that is that nothing has happened" (a saying which illustrates that negative observations can be important). Yet we should think twice before making such an assumption, since there are many examples that show the major hazards of a deterministic model.

Real-world optimisation problems are often subject to uncertainty: measurement errors, implementation errors, system disturbances, or incomplete available data at the moment of planning. All of these factors can make the nominal solution completely meaningless from a practical point of view, and there is hardly any real context which is immune from these dangers. For this reason, many approaches to deal with uncertainty have been developed. Among them, *robust optimisation* has received a consistent share of attention from the research community over the last twenty years [1, 5].

More recently, the limitations of "classical" robustness, in terms of "conservativeness" of the solutions, has lead to extensions towards robust multi-stage models: *adjustable robustness* [2] and *recoverable robustness* [9], respectively designed for continuous and discrete settings. In both cases, second-stage decisions, called *recovery*, are allowed.

In this paper, moving from the recoverable robust paradigm, we present a framework to hedge against *delay propagation* in *event-based* problems. These are problems featuring time-related activities, called *events*, having some properties that we need to decide upon, namely, roughly speaking, "when" and/or "where" they take place. In other words, the (nominal)

*DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy. E-mail: {alberto.caprara, l.galli, paolo.toth}@unibo.it

†Institut für Mathematik, Technische Universität Berlin, Straße des 17. Juni, 136 10623 Berlin, Germany. E-mail: stiller@math.tu-berlin.de

optimisation problem consists of deciding upon the property values of each event according to some objective function and constraints.

We use a network to model the delay-robust counterpart of the nominal problem. Each event corresponds to a node in the network and the arcs model possible *precedence relationships* among them, thus they define how the delay propagates from one event to another. The arcs are *buffered* in the sense that they may absorb delay. The corresponding buffer values are uniquely determined by the property values of the events. In other words, the network structure is not given "a priori", but it is implicitly defined by the choices on the event properties. The scenarios are associated with disturbances over the time instants (i.e., delays with respect to the nominal times of events). The only recovery action consists of propagating the delay over the given network, shifting forward the events in time. Hence, the delay-robust version of the (nominal) optimisation problem consists of assigning the property values to each event so that the corresponding network minimises the total delay propagation.

Moving from theory to practice, our aim in this paper is to present a general mathematical framework and evaluate its effectiveness in a real-life context. 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 of assigning trains to platforms in a railway station and defining the corresponding routes (arrival and departure paths) in the station area [6], and is a key problem in railway planning [8].

Apart from a large penalty for those trains not assigned to any platform (or, to use the terminology in [6], assigned to a dummy platform), an important term in the nominal TPP objective function used in [6] was a penalty assigned to train pairs occupying the same resource "close" in time. Such a term was a rough way to prevent delay propagation. In our experiments, we prove that our framework can be applied to real-world instances and the resulting robust solutions have significantly less delay propagation than those optimised for the nominal objective above, i.e., the objective of [6].

The structure of the paper is as follows. In the reminder of introduction we review the relevant literature. In Section 1, we discuss our mathematical framework to deal with delay-robust event networks, which is applied to TPP in Section 2, where we discuss how we solved it. Finally, in Section 3, we tackle real-world TPP instances provided by Rete Ferroviaria Italiana (RFI, the main Italian Railway Infrastructure Manager) to show the practical effectiveness of our approach.

Robustness Issues in the Literature

The framework we present in this paper is based on the *recoverable robust* paradigm and moves from the so called *network buffering* approach, both presented in Liebchen *et al.* [9]. Recoverable robustness integrates *classical robust optimisation* and *2-stage stochastic programming*.

The now classical notion of robust optimisation was first introduced by Soyster [10]. The point is to find a *unique* solution that is feasible for all possible scenarios. In [10] it is shown how a *Linear Program* (LP) with convex column-wise uncertainty in the coefficients of the constraint matrix could be handled via another LP. The approach is too conservative to be used in practice, though.

The real development of robust optimisation is due to Ben-Tal and Nemirovski [3, 4], who presented a model for LPs with coefficient uncertainty trying to limit conservatism via ellipsoidal uncertainty sets. Clearly, the drawback is the necessity to solve non-linear programs, which makes it practically doable only for continuous problems.

For this reason Bersimas and Sim [5] proposed an alternative robust LP framework characterised by *budget* uncertainty sets. Each coefficient can take a value in a symmetric interval around a nominal value. The budget parameter is used to limit the number of coefficients that can simultaneously take their worst-case value for each constraint. Surprisingly, this row-wise budget uncertainty model retains linearity. Hence their framework is particularly appealing for a discrete setting.

Discrete optimisation problems are generally difficult to solve and the introduction of additional complexity (e.g., non-linearities) in the robust counterpart should be avoided. Some authors have studied specific robust versions for classical discrete problems, but they have not proposed any general framework. For this reason, for many real-world (and large-scale) discrete problems (e.g., railway planning) robust optimisation has been kept out of sight.

Another reason is that, although classical robust optimisation may appeal for its performance guarantee (a feasible solution has to be feasible for all likely scenarios), it is generally too expensive for practical contexts. In fact, such "one-fits-all" solutions are too conservative in a number of applications. In particular, this approach does not adequately model the requirement for optimisation under uncertainty found in many real-world settings. Certain limited adjustments to a solution during operations are common practice and a model that does not exploit these possibilities cannot deliver the desired solutions.

This difficulty is overcome in two-stage stochastic integer programming models. In such models, initial (first-stage) decisions are planned prior to the realisation of uncertain parameters, whereas second-stage decisions provide recourse opportunities given a realisation.

Similar in spirit to two-stage stochastic programming models are the adjustable robust approach for LP developed in Ben-Tal *et al.* [2] and the recoverable robust paradigm by Liebchen *et al.* [9]. The former moves from previous works of Ben-Tal and Nemirovski, thus it is more suited for continuous settings; the latter, thanks to its generality, is also applicable to discrete settings.

In this paper we consider a class of problems aimed at scheduling events with time-distance requirements. From a robust optimisation point of view, such a network can be viewed as a way to model delay propagation among the events. The scheduling should include buffer times to absorb delays, which otherwise would spread through. A good schedule concentrates the buffers on the neuralgic points allowing limited worst-case propagation. The problem of distributing buffers in a given network to minimise the maximal total delay that can be caused by initial disturbances is known as the robust *network buffering* problem. This problem is a special case of recoverable robustness as explained in [9, 11]. The case considered here is more complex, as the definition of the buffers is not direct but implicit in the definition of the nominal solution.

1 Delay-Robust Event Scheduling: a Framework

In this section we illustrate a general framework approach for delay-robust event scheduling, where the robust part of the objective function calls for the minimisation of delay propagation. Formally, the framework can be defined as follows. We are given:

- A set $\mathcal{E}$ of *events* to be scheduled, i.e., for which we have to define when and where they will take place.
- A *nominal problem* of the form $\min\{c(x) : x \in F\}$, aimed at scheduling the events so as to minimise a suitable nominal objective function $c : F \to \mathbb{R}$.

- A *delay-propagation network* $N = (\mathcal{E}, A)$, with $(e', e) \in A$ if a delay $d_{e'}$ on event e' may propagate to a delay d_e on event e, according to the relation $d_e \geq d_{e'} - b((e', e), x)$, where $b((e', e), x)$ is a *buffer time* function $b : A \times F \to \mathbb{R}_+$, depending on the scheduling of the two events in the solution $x \in F$.

- A set $\mathcal{S}$ of possible *scenarios*, where each scenario $s \in \mathcal{S}$ is defined by external *disturbance* $\delta_e^s \geq 0$ assigned to each event $e \in \mathcal{E}$ and is represented by vector $\delta^s \in \mathbb{R}_+^{|\mathcal{E}|}$.

Note that buffer times model the absorption of delays and that $b((e', e), x)$ is infinite in case the events e' and e do not interact in the solution x. Note also that N is not necessarily acyclic, i.e. the order of the events may not be fixed a priori. Finally, note that $\mathcal{S}$ is not necessarily finite nor countable.

Let d_e^s denote the delay of event e for scenario s, which depends on the buffer times (and hence on the solution $x \in F$). The *cumulative delay* over all events associated with scenario s is then defined by $\sum_{e \in \mathcal{E}} d_e^s$.

The associated *delay-robust problem* is the bi-objective optimisation problem calling for the minimisation of the nominal objective function $c(x)$, and, at the same time, of the maximum cumulative delay over all scenarios in $\mathcal{S}$. For concreteness, let us focus on the case in which, after appropriate scaling, the two objectives are summed together into a unique objective function. The delay-robust problem can then be formulated as:

$$\min c(x) + D, \tag{1}$$

subject to

$$x \in F, \tag{2}$$

$$D \geq \sum_{e \in \mathcal{E}} d_e^s, \qquad s \in \mathcal{S}, \tag{3}$$

$$d_e^s \geq \delta_e^s, \qquad e \in \mathcal{E},\ s \in \mathcal{S}, \tag{4}$$

$$d_e^s \geq d_{e'}^s - f_{(e',e)}, \qquad (e', e) \in A,\ s \in \mathcal{S}, \tag{5}$$

$$f_{(e',e)} = b((e', e), x), \qquad (e', e) \in A. \tag{6}$$

Here, the variables are x, the nominal problem ones, D, denoting the maximum cumulative delay over all scenarios, d_e^s, denoting the delay of event e for scenario s and solution x, and $f_{(e',e)}$, denoting the buffer time of arc (e', e) for solution x. Note in particular that the buffer times do not depend on the scenario. Constraints (3) define the value of variable D, while constraints (4) and (5) express the fact that the delay of event e is the maximum between the external disturbance δ_e^s and the delay propagated from each event e' such that $(e', e) \in A$, which is partly absorbed by buffer $f_{(e',e)}$.

In the following we will discuss on concrete examples of the scenario set $\mathcal{S}$ and the buffer time function $b(\cdot)$ that are of interest in practice and that allow the practical solution of the very general formulation above.

1.1 Bounded total disturbances as scenario set

In the spirit of previous works on robustness reviewed in the introduction, also here we impose the natural condition that, for any scenario $s \in S$, the total amount of external disturbances

be bounded. In our case, this is formalised by $\sum_{e\in\mathcal{E}} \delta_e^s \le \Delta$, for a given parameter Δ. In other words, the set of scenarios is defined by the vectors:

$$\{\delta^s \in \mathbb{R}_+^{|\mathcal{E}|} : \|\delta^s\|_1 \le \Delta\} \tag{7}$$

These are *uncountably* many scenarios, which can however be handled easily thanks to the following observation.

Proposition 1 *For the delay-robust problem* (1)-(6)*, the* uncountable *scenario set defined by* (7) *is equivalent to the* finite *scenario set S defined by the vectors*

$$\{\delta^s \in \{0,\Delta\}^{|\mathcal{E}|} : \|\delta^s\|_1 = \Delta\}, \tag{8}$$

which contains only $|\mathcal{E}|$ scenarios.

Proof. For a fixed $x \in F$ and corresponding buffers $f_{(e',e)}$ defined by (6), the value of D is determined by the maximum over all scenarios of the LP calling for the minimisation of $\sum_{e\in\mathcal{E}} d_e^s$ subject to (4) and (5). These are uncountably many LPs which differ by the right-hand-sides, having the form $\begin{bmatrix} \delta^s \\ -f \end{bmatrix}$ for s in (7), i.e., their right-hand-sides belong to a polytope. It is well known and easy to verify that the maximum value over all these LPs is then attained for a right-hand-side which is a vertex of the polytope, and (modulo translation by f) that the vertices of this polytope are given by (8). This shows that, for every $x \in F$, one may restrict attention to the scenarios in (8), proving the claim. □

In other words, one has to consider only the $|\mathcal{E}|$ scenarios in which the maximum disturbance Δ is applied to *one single* event. Note that this does not assume any structure on the nominal objective function $c(\cdot)$ and feasible region F. In the following we denote by s not only a generic scenario, but also the event subject to the disturbance Δ.

1.2 Quadratic buffer time function

In the most natural case, the buffer time $b((e',e),x)$ only depends on the way in which events e' and e are scheduled in x, and is independent of the scheduling of the other events. Let $\mathcal{L}_e$ denote the set of possible schedules for each event $e \in \mathcal{E}$, and $\varphi(e',\ell',e,\ell)$ denote the value of the buffer time in case e' has schedule $\ell' \in \mathcal{L}_{e'}$ and e has schedule $\ell \in \mathcal{L}_e$.

Assume that x contains binary variables $x_{e,\ell}$ equal to 1 if event e has schedule $\ell \in \mathcal{L}$. (Even if not present explicitly in x, these variables are a linear function of the existing ones, see for instance our application.) The explicit form of constraints (6) is then:

$$f_{(e',e)} = \sum_{\ell'\in\mathcal{L}_{e'}} \sum_{\ell\in\mathcal{L}_e} \varphi(e',\ell',e,\ell)\; x_{e',\ell'}\; x_{e,\ell}, \qquad (e',e) \in A. \tag{9}$$

These quadratic constraints can be linearised in a few ways. Among these, the most convenient one appears to be the method discussed in [6, 7], which avoids the explicit introduction of binary variables to model the product $x_{e',\ell'}\; x_{e,\ell}$.

Proposition 2 *It is possible to linearise* (9) *without introducing additional variables, using a set of linear inequalities that can be separated by solving an LP with $|\mathcal{L}_{e'}||\mathcal{L}_e|$ variables and $1 + |\mathcal{L}_{e'}| + |\mathcal{L}_e|$ constraints.*

Proof. The method in [6, 7] uses the fact that the $x_{e,\ell}$ variables must satisfy the constraints

$$\sum_{\ell \in \mathcal{L}_e} x_{e,\ell} = 1, \qquad e \in \mathcal{E}. \tag{10}$$

Accordingly, without introducing additional variables, (9) can be linearised by constraints of the form

$$f_{(e',e)} \leq \sum_{\ell' \in \mathcal{L}_{e'}} \alpha_{e',\ell'}\ x_{e',\ell'}\ + \sum_{\ell \in \mathcal{L}_e} \alpha_{e,\ell}\ x_{e,\ell} + \beta_{(e,e')}, \tag{11}$$

which are valid inequalities for the convex hull of all vectors $(f_{(e',e)}, (x_{e',\ell'}), (x_{e,\ell})) \in \mathbb{R} \times \{0,1\}^{|\mathcal{L}_{e'}|} \times \{0,1\}^{|\mathcal{L}_e|}$ that satisfy (9) and (10). Given that there are only $|\mathcal{L}_{e'}||\mathcal{L}_e|$ such vectors, separation of (11) can be carried out by testing explicitly if the given solution is a convex combination of these vectors, by solving an LP of the size given in the statement. To conclude the proof, note that we only need inequalities that upper bound $f_{(e',e)}$ for our purposes, so we can restrict attention to (11). □

For details, see [6, 7].

1.3 Combinatorial Benders decomposition

Note that, even if the scenario set is limited to (8), there are $|\mathcal{E}|^2$ delay variables d_e^s and $|\mathcal{E}||A|$ constraints (5), making formulation (1)-(6) very large. On the other hand, by using Benders decomposition, one may directly optimise over the feasible region obtained by projecting out the d_e^s variables (and removing the constraints (3)-(5)), as follows.

Let $\sigma_{\overline{f}}(s,e)$ be the value of the shortest path from s to e on the delay propagation network $N = (\mathcal{E}, A)$ with arc lengths $\overline{f}_{(e',e)}$ for $(e',e) \in A$. Moreover, let $P_{\overline{f}}(s,e) \subseteq A$ denote such a shortest path.

Lemma 1 *Given buffer time values $\overline{f}_{(e',e)}$, an external disturbance equal to Δ on event $s \in \mathcal{E}$ (and null for the other events) propagates to a delay of value $\max\{0, \Delta - \sigma_{\overline{f}}(s,e)\}$ on event $e \in \mathcal{E}$.*

Proof. Recall that the d_e^s variables attain the minimum possible value due to the objective function (1). For $\delta_s^s = \Delta$ and $\delta_e^s = 0$ for $e \neq s$, inequalities (4) imply $d_s^s = \Delta$ and the non-negativity of the other d_e^s, whereas inequalities (5) imply that d_e^s is equal to Δ plus the value of the longest path from s to e with arc lengths $-\overline{f}_{(e',e)}$. Changing sign to the arc lengths yields the claim. □

A direct consequence is:

Lemma 2 *Given buffer time values $\overline{f}_{(e',e)}$ and maximum cumulative delay $\overline{D}$, for every scenario $s \in \mathcal{S}$ there exist values of the d_e^s variables satisfying (3)-(5) if and only if $\sum_{e \in \mathcal{E}} \max\{0, \Delta - \sigma_{\overline{f}}(s,e)\} \leq \overline{D}$. Otherwise, letting $\mathcal{E}_{\overline{f}}(s) \subseteq \mathcal{E}$ be the set of events such that $\Delta - \sigma_{\overline{f}}(s,e) > 0$, a valid inequality that is violated by $\overline{f}_{(e',e)}$ and $\overline{D}$ is*

$$\sum_{e \in \mathcal{E}_{\overline{f}}(s)} \sum_{a \in P_{\overline{f}}(s,e)} f_a \leq D. \tag{12}$$

Note that some arc $a \in A$ may appear in more than one path in (12) and therefore the associated variable f_a may have coefficient greater than 1. Lemma 2 immediately implies:

Proposition 3 *One may separate over the projection of the set of feasible solutions of* (3)-(5) *onto the space of the* $f_{(e',e)}$ *and* D *variables by computing* $|\mathcal{E}|$ *shortest paths in* N *with non-negative edge lengths.*

For instance, separation can be carried out by using Dijkstra's algorithm.

2 Delay-Robust Train Platforming

In TPP, one is given a major railway station and a set T of trains whose arrival and departure times and entry and exit points at the station are specified. TPP calls for assigning each of these trains a *stopping platform*, an *arrival path* from its entry point to the platform, and a *departure path* from the platform to its exit point.

In the application we consider, an explicit list of arrival and departure paths is given, along with the associated *occupation time*, i.e., the time taken to travel along them (possibly dependent on the train), and a list of incompatible path pairs, representing the fact that the two paths physically intersect each other. Side constraints impose that no two trains occupy incompatible paths for a time window of duration larger than a so-called *conflict threshold*. Moreover, side constraints also impose that no two trains are simultaneously present at the same platform and, after a train leaves the platform, a minimum so-called *headway time* (possibly dependent on the platform and the two trains considered) must elapse before another train arrives at the platform.

2.1 Events and delay-propagation network

According to the above discussion, for each train $t \in T$ the nominal solution defines (i) the instant in which t occupies (i.e., starts to occupy) its arrival path, (ii) the instant in which t frees (i.e., ends to occupy) its arrival path, which coincides with the instant in which t occupies its platform, (iii) the instant in which t frees its platform, which coincides with the instant in which t occupies its departure path, and (iv) the instant in which t frees its departure path. All these "events" in principle may be subject to external disturbances, which however in practice occur essentially only on two of them, namely (i), in case the train arrives late at the station, and (iii), in case the platform operations take longer than required. In other words, it is widely realistic to assume that the travel time along the arrival and departure paths is not affected by external disturbances. Accordingly, to model TPP within our framework, the following two events are associated with each train $t \in T$: *arrival* a_t, meaning that the train occupies a given arrival path at a given instant, and *departure* p_t, meaning that the train frees its platform at a given instant. This defines $\mathcal{E}$.

In the most general TPP setting, for the scheduling of a_t one has to define the instant, the arrival path, the platform, and the travel time along the path (which determines the arrival time at the platform). For the scheduling of p_t one has to define the instant, the platform, the departure path, and the travel time along the path (which determines the instant at which the path is freed).

In the associated *delay-propagation network* $N = (\mathcal{E}, A)$, there are the following arcs in A joining events of the same train $t \in T$:

- (a_t, p_t), meaning that a delay in a_t may cause a delay in p_t, depending on the travel time along the arrival path and the stopping time at the platform in the nominal solution and the minimum possible values of these two times.

Moreover, there are the following arcs in A joining events of two trains $t', t \in T$:

- $(a_{t'}, a_t)$, meaning that a delay for t' in occupying its arrival path may cause a delay for t in occupying its arrival path, in case the arrival times of t' and t are sufficiently close and t' and t are scheduled on incompatible arrival paths;
- $(a_{t'}, p_t)$, meaning that a delay for t' in occupying its arrival path may cause a delay for t in freeing its platform, in case the arrival time of t' and the departure time of t are sufficiently close and t' and t are scheduled on incompatible arrival and departure paths, respectively;
- $(p_{t'}, a_t)$, meaning that a delay for t' in freeing its platform may cause a delay for t in occupying its arrival path, in case the departure time of t' and the arrival time of t are sufficiently close and t' and t are either scheduled on the same platform, or on incompatible departure and arrival paths, respectively;
- $(p_{t'}, p_t)$, meaning that a delay for t' in freeing its platform may cause a delay for t in freeing its platform, in case the departure times of t' and t are sufficiently close and t' and t are scheduled on incompatible departure paths.

As already mentioned, the values of the buffer time for each arc joining the events of two distinct trains is uniquely defined by the way in which these two events are scheduled in the nominal solution.

Example 1 *Assume in the nominal solution trains t' and t stop at the same platform, event $p_{t'}$ is scheduled at 10:00, and event a_t at 10:04 with 3 minutes to travel along the arrival path, so that t occupies the platform at 10:07. Moreover, assume that the headway time that must elapse between t' freeing the platform and t occupying it is 2 minutes and that the 3-minute travel time for t' along its arrival path is fixed. If the departure path of t' and the arrival path of t are compatible, the buffer time $f_{(p_{t'}, a_t)}$ is equal to 5 minutes, as a delay of up to 5 minutes on $p_{t'}$ does not affect a_t, whereas a larger one does.*

2.2 Getting rid of the nominal objective function

The nominal objective function considered in [6] was derived after long discussion with the practitioners and had quite a few terms. However, as widely discussed in the experiments in [6] and mentioned in the introduction, only two of these terms play a relevant role in the optimisation.

The first term is a very large penalty for the use of "dummy" platforms, which do not exist, and for the assignment of trains to these platforms. Two trains cannot occupy simultaneously the same dummy platform and this may require the use of more than one such platforms. It is so important to optimise this term that no robustness issue would justify increasing it with respect to the nominal optimum, so we had to pre-process our instances by minimising it and by removing the trains assigned to dummy platforms from the delay-robust version of TPP.

The second term is a penalty for two trains occupying incompatible paths for a time window of duration not larger than the conflict threshold. In fact, this is a rough way to try to limit delay propagation, mainly caused by such simultaneous occupations, which can be abandoned as soon as we cast our problem in terms of delay-robust event scheduling. Accordingly, for TPP we focus on the minimisation of the total cumulative delay D assuming $c(\cdot)$ identically null.

2.3 The overall delay-robust model and its solution

As in [6], we represent the nominal solutions by associating *patterns* with trains. Specifically, for each train $t \in T$ there is a collection $\mathcal{P}_t$ of patterns, each representing a feasible scheduling of both events a_t and p_t illustrated above. This has the advantage of encoding within the notion of patterns the constraints that relate these two events. At the same time, at least for our real-world application, there is a relatively small number of patterns that can be constructed and stored explicitly without resorting to column generation. Instead, we use *pricing*, solving LP relaxations with a small subset of the variables and computing the reduced costs of all other variables to check if some have to be added.

The side constraints relating the patterns are then imposed as generic incompatibilities between pattern pairs, defining an associated *pattern-incompatibility* graph. As to the definition of the buffer times via the counterpart of (9), it can be done by introducing in place of $\varphi(e', \ell', e, \ell)$ the value $\varphi(e', P', e, P)$, which is the value of the buffer time $b((e', e), x)$ in case events e' and e are scheduled respectively according to patterns P' and P.

The overall delay-robust problem can be formulated by using the following variables. For each $t \in T$ and $P \in \mathcal{P}_t$, variable $x_{t,P}$ indicates whether train t is assigned pattern P. Moreover, for each $t \in T$ and $s \in \mathcal{S}$ let $d^s_{a_t}$ and $d^s_{p_t}$ be the two continuous variables expressing the delay of events a_t and p_t in scenario s.

Letting $\mathcal{K}$ denote the collection of the maximal cliques of the pattern-incompatibility graph, the formulation reads:

$$\min D \tag{13}$$

subject to

$$\sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{14}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \tag{15}$$

$$x_{t,P} \in \{0,1\}, \qquad t \in T,\ P \in \mathcal{P}_t, \tag{16}$$

$$f_{(e',e)} = \sum_{P' \in \mathcal{P}_{t'}} \sum_{P \in \mathcal{P}_t} \varphi(e', P', e, P)\, x_{t',P'}\, x_{t,P}, \qquad t', t \in T,\ e' \in \{a_{t'}, p_{t'}\},\ e \in \{a_t, p_t\},\ (e', e) \in A, \tag{17}$$

and to (3)-(5). The linearisation of the quadratic constraints (17) is fully identical to the one of (9) via (11) discussed in Section 1.2. The resulting linearised constraints have the form:

$$f_{(e',e)} \leq \sum_{P' \in \mathcal{P}_{t'}} \alpha_{e',P'}\, x_{e',P'} + \sum_{P \in \mathcal{P}_t} \alpha_{e,P}\, x_{e,P} + \beta_{(e,e')}. \tag{18}$$

With such a linearisation, the overall model is a *Mixed-Integer Linear Program* (MILP).

How to handle the clique constraints (15), which are exponentially many and NP-complete to separate, is discussed in [6]. In brief, the point is to impose explicitly a small subset of these constraints related with patterns occupying the same platform at the same time, and to dynamically separate a suitable relaxation of the remaining ones (which suffices to define a valid MILP model). This relaxation considers only patterns associated with two trains, and the associated constraints can be separated efficiently by finding cliques of maximum weight in bipartite graphs.

Our solution method is the branch-and-bound algorithm proposed in [6], based on the pricing of the $x_{t,P}$ variables and the separation of constraints (15). Branching is aimed at

quickly finding a good feasible solution, by selecting the fractional $x_{t,P}$ variable with highest cost and exploring first the branch in which it is fixed to 1.

In addition to the method of [6], here we also added the separation (18), discussed in Section 1.2, and the possible Benders decomposition of (3)-(5), discussed in Section 1.3. Moreover, we also added the following fast constructive heuristic, applied every time we solve the current LP relaxation. The heuristic considers the variables $x_{t,P}$ by decreasing values in the current LP solution (the value being 0 for the variables not in the current LP), breaking ties for the variables having value 0 by increasing reduced costs. Following the order, each variable is fixed to 1 if this is compatible with all variables already fixed, and to 0 otherwise. Given that we can consider explicitly all patterns, this heuristic always produced a feasible solution for the instances in our benchmark.

3 Experimental Study

In this section we discuss the application of our delay-robust approach to a real-world TPP case. We show that, if the model is properly handled taking into account its large size, the resulting solutions have a notably smaller delay propagation than the nominal one.

3.1 The benchmark instances

The instances of our case study come from Rete Ferroviaria Italiana. This is the same benchmark data used in the deterministic study reported in [6], which considers the stations of Palermo Centrale, Genova Piazza Principe, Bari Centrale and Milano Centrale. As discussed in Section 2.2, we first pre-process the instances by minimising the cost related with the use of "dummy" platforms, and removing the associated trains. We then solve the delay-robust problem for the remaining trains, which are, respectively, 137 for Palermo Centrale, 91 for Genova Piazza Principe, 196 for Bari Centrale, and 118 for Milano Centrale.

instance	station name	$\|T\|$	$\|B\|$	$\|D\|$	$\|R\|$	# inc.
PA C.LE.	Palermo Centrale	137	11	4	64	1182
GE P.PR.	Genova Piazza Principe	91	10	4	174	7154
BA C.LE.	Bari Centrale	196	14	5	89	1996
MI C.LE.	Milano Centrale	1818	24	7	312	29294

Table 1: Instances in our case study.

Table 1 summarizes the characteristics of the instances used in our case study, reporting the instance name, the full name of the corresponding station, the numbers of trains ($|T|$), platforms ($|B|$), directions ($|D|$), and paths ($|R|$) and the number of pairs of incompatible paths (*# inc.*).

The large size of the delay-robust problem, as opposed to its nominal counterpart, forces us to naturally decompose these instances into subinstances associated with "time windows" during the day, considering the trains in increasing order of arrival time and restricting attention to consecutive trains according to this order. The subinstances are defined by equally splitting the trains among them, i.e., if we consider w subinstances and $|T|$ is a multiple of w, we have a first subinstance with the first $|T|/w$ trains, a second subinstance with the next $|T|/w$ trains, and so on. Specifically, for Palermo Centrale we have 4 subinstances each with 34 or 35

trains, for Genova Piazza Principe we have 4 subinstances each with 22 or 23 trains, for Bari Centrale we have 8 subinstances each with 24 or 25 trains, and for Milano Centrale we have 6 subinstances each with 19 or 20 trains.

The scenarios are defined by (8) with $\Delta = 30$ minutes, i.e., each one corresponds to a delay of 30 minutes on the arrival of a train at the station or on the departure of a train from its platform.

We finally remark that for our instances, independently of the event scheduling, the buffer times are null for all arcs (a_t, p_t) for $t \in T$, given that in the nominal solution the arrival time at the platform is always equal to the arrival time at the station plus the travel time along the arrival path, and the stopping time at the platform is the minimum possible.

3.2 Implementation details

Our method was implemented in ANSI C and tested on a PC Pentium 4, 3.2 GHz, with a 2 GB RAM. We used the callable library CPLEX 10.0 to solve the LP relaxations, and implemented all other parts ourselves.

Notice that the constraints (18) that we separate are valid *locally*, i.e., for the subset of the $x_{t,P}$ variables that are currently in the model. This means that when a new $x_{t,P}$ variable is added by the pricing procedure, we need to update all the buffer constraints according to the coefficients calculated in the pricing phase. On the other hand, the constraints (15) are valid *globally*, and it would not be strictly necessary to update them during the pricing procedure, but still we do it by maintaining the cliques *maximal* in order to get a stronger LP relaxation.

As to Benders decomposition, illustrated in Section 1.3, extensive computational results have shown that it is much slower than the "compact" formulation with the explicit presence of the d_e^s variables and constraints (3)-(5). In fact, the best option is to dynamically separate also these constraints, by direct enumeration, introducing the corresponding d_e^s variables only when needed.

3.3 Computational results

In Table 2 we report the results for the 22 instances mentioned above. For the "nominal" solution found by the method proposed in [6], in which the objective function is a rough way to limit delay propagation as discussed in Section 2.2, and for the LP relaxation of the MILP formulation of the delay-robust model, we report the total propagated delay D and the associated computing time (*time*) in seconds. Finally for the best solution found by the delay-robust approach, we report the total propagated delay D, the percentage gap (*%gap*) with respect to the LP value, the percentage reduction (*%reduction*) with respect to the solution found by the method proposed in [6], and the computing time (*time*) in seconds. The values of the propagated delay D are expressed in minutes.

The results show that, although the LP relaxation appears to be fairly weak, as the associated gaps with respect to the integer solutions are large, and the computing times of the delay-robust approach are non-negligible, the reduction of the total propagated delay with respect to the nominal solution is significant, ranging between 12% and 29% for Palermo Centrale, between 19% and 35% for Genova Piazza Principe, between 0% and 23% for Bari Centrale, and between 1% and 44% for Milano Centrale, with an average of 17%. This shows the effectiveness of the delay-robust approach for our TPP case.

instance	# trains	[6] Solution D	time	Delay-Robust LP D	time	Delay-Robust Solution D	%gap	%reduction	time
PA I	34	1112	5	451.88	131	791	43%	29%	9788
PA II	34	694	5	252.00	49	584	57%	16%	3892
PA III	34	756	5	285.00	47	593	52%	22%	5411
PA IV	35	525	5	207.00	48	459	55%	13%	1417
GE I	22	561	5	193.11	30	453	57%	19%	147
GE II	23	1252	5	388.00	201	810	52%	35%	30417
GE III	23	1073	5	297.95	118	865	66%	19%	1086
GE IV	23	758	5	231.00	54	498	54%	34%	274
BA I	24	770	5	341.00	68	592	42%	23%	1095
BA II	24	959	5	508.08	118	959	47%	0%	2
BA III	24	711	5	293.91	115	651	55%	8%	525
BA IV	24	733	5	308.40	103	686	55%	6%	2431
BA V	25	786	5	245.26	165	734	67%	7%	3204
BA VI	25	819	5	312.00	51	819	62%	0%	2
BA VII	25	1025	5	315.18	77	743	58%	27%	4126
BA VIII	25	714	5	240.00	45	580	59%	19%	535
MI I	19	480	5	354.00	105	476	26%	1%	201
MI II	19	785	5	449.28	330	686	35%	13%	2415
MI III	20	1102	6	382.88	334	822	53%	25%	938
MI IV	20	1067	5	721.00	839	951	24%	11%	2547
MI V	20	1002	8	290.76	288	565	49%	44%	2503
MI VI	20	898	5	415.13	655	816	49%	9%	618

Table 2: Results for the real-world instances of Rete Ferroviaria Italiana.

Conclusions

We proposed a framework to compute the delay-robust counterpart of event-based optimisation problems. Such a framework consists of the original (nominal) optimisation problem stitch up with a delay-propagation network. The generality and simplicity of the delay-propagation model allows one to attach it to the nominal formulation together with linking constraints to translate nominal solutions into network buffers.

Further, we tested our framework on a real-world railway planning problem, namely train platforming. For real-world instances from the main Italian Infrastructure Manager, the new method finds solutions in which the total propagated delay is significantly smaller than the one for the previous "nominal" solutions, although the dominant term in the objective function of the latter was also aimed at reducing delay propagation.

Acknowledgment

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

References

[1] A. Ben-Tal, L. El Ghaoui and A. Nemirovski (2009). *Robust Optimization.* Princeton University Press.

[2] A. Ben-Tal, A. Goryashko, E. Guslitzer and A. Nemirovski (2004). Adjustable Robust Solutions of Uncertain Linear Programs. *Math. Progr.*, 99, 351–376.

[3] A. Ben-Tal and A. Nemirovski (1998). Robust Convex Optimization. *Math. of Oper. Res.*, 23, 769–805.

[4] A. Ben-Tal and A. Nemirovski (1999). Robust Solutions of Uncertain Linear Programs. *OR Letters*, 25, 1–13.

[5] D. Bertsimas and M. Sim (2004). The Price of Robustness. *Oper. Res.*, 52, 35–53.

[6] A. Caprara, L. Galli and P. Toth (2011). Solution to the Train Platforming Problem. *Tran. Sci.*, 45(2), 246–257.

[7] A. Caprara, L. Galli and P. Toth (2010). A New Linearisation Method for Quadratic Semi-Assignment. Working paper, DEIS, Universtà di Bologna.

[8] A. Caprara, L. Kroon, M. Monaci, M. Peeters and P. Toth (2007). Passenger Railway Optimization. In C. Barnhart, G. Laporte (eds.), *Transportation.* Handbooks in Operations Research and Management Science 14, Elsevier, 129–187.

[9] C. Liebchen, M. Lübbecke, R.H. Möhring and S.Stiller (2009). The Concept of Recoverable Robustness, Linear Programming Recovery, and Railway Applications. In R.K. Ahuja, R.H. Möhring and C.D. Zaroliagis (eds.), *Robust and On-Line Large Scale Optimization.* Lecture Notes in Computer Science 5868, Springer-Verlag, 1–27.

[10] A.L. Soyster (1973). Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming. *Oper. Res.*, 1154–1157.

[11] S. Stiller (2008). *Extending Concepts of Reliability* PhD thesis, Institut für Mathematik, Technische Universität Berlin, Berlin, Germany.