Alma Mater Studiorum Università di Bologna

Dottorato di Ricerca in
Automatica e Ricerca Operativa

MAT/09

XXI Ciclo

Combinatorial and Robust Optimisation Models and Algorithms for Railway Applications

Laura Galli

Il Coordinatore
Prof. Claudio Melchiorri

I Relatori
Prof. Paolo Toth
Prof. Alberto Caprara

Esame finale 2009

Contents

Acknowledgments

There are a number of people who were instrumental in helping me through my studies. I am extremely grateful to my supervisors and all those who have been collaborating with me.

Bologna, March 5, 2009

Laura Galli.

Keywords

Combinatorial Optimisation, Integer Programming, 0-1 Quadratic Programming, Train Platforming, Robust Optimisation, Recoverable Robustness, Network Buffering, Online Planning, Rescheduling, Rolling Stock Planning.

Abstract

This thesis deals with an investigation of *combinatorial and robust optimisation models* to solve *railway problems*. Railway applications represent a challenging area for *operations research*. In fact, most problems in this context can be modelled as combinatorial optimisation problems, in which the number of feasible solutions is finite. Yet, despite the astonishing success in the field of combinatorial optimisation, the current state of algorithmic research faces severe difficulties with highly-complex and data-intensive applications such as those dealing with optimisation issues in large-scale transportation networks.

One of the main issues concerns *imperfect information*. The idea of *Robust Optimisation*, as a way to represent and handle mathematically systems with not precisely known data, dates back to 1970s. Unfortunately, none of those techniques proved to be successfully applicable in one of the most complex and largest in scale (transportation) settings: that of railway systems. Railway optimisation deals with planning and scheduling problems over several time horizons. Disturbances are inevitable and severely affect the planning process. Here we focus on two compelling aspects of planning: *robust planning* and *online (real-time) planning*.

The thesis is organised as follows:

Chapter 1 The *Introduction* contains a detailed overview of the thesis.

Chapter 2 We describe the main case study, that of *Train Platforming*, for which we present an Integer Programming (IP) formulation together with an exact solution approach.

Chapter 3 We introduce a new paradigm for Robust Optimisation, namely *Recoverable Robustness*.

Chapter 4 We investigate the concept of *Price of Recoverability*, formulated within the context of Recoverable Robustness, in a real-world setting, that of *Railway Rolling Stock Planning*.

Chapter 5 We investigate the concept of Recoverable Robustness via *Network Buffering* for the Train Platforming Problem.

Chapter 6 We describe an on-going research for *online (real-time) Rescheduling*, applied to the Train Platforming Problem.

List of Figures

List of Tables

"And now you know the words," she added, as she put her head down on Alice's other shoulder, "just sing it through to me."

Through the Looking Glass, Lewis Carroll.

Chapter 1

Introduction

Tucked away in the very far corners of South West Cornwall is the little town of Camborne. Packed full of stunningly beautiful places, ravishing surrounding areas, rivers, creeks, beaches, Camborne is a friendly, warm and hospitable village that is every bit as charming as it looks. Still, this village represents a geographical milestone, not for its beauty, but for its history. In fact, it is believed to be the first town in the world that witnessed wagons hauled by a real steam locomotive.

1.1 The birth of the railway

The first full scale railway steam locomotive was built in 1804 by Richard Trevithick, an English engineer born in Cornwall. He built a full-size steam road locomotive on a site near the present day Fore Street at Camborne. He named the carriage "Puffing Devil" and, on Christmas Eve that year, he demonstrated it by successfully carrying several men up Fore Street and then continuing on up Camborne Hill to the nearby village of Beacon. This event is widely recognised as the first demonstration of transportation powered by steam, and it later inspired the popular Cornish folk song "Camborne Hill". During further tests, Trevithick's locomotive broke down three days later, after passing over a gully in the road. The vehicle was left under some shelter with the fire still burning. Meanwhile the water boiled off, the engine overheated and the whole machine burnt out, completely destroying it. Trevithick however did not consider this episode a serious setback but more a case of operator error. Indeed, railway transportation had just started its successful journey in history, and was not to stop. Trevithick built a new steam locomotive and this drew five wagons passengers and a coach with 70 passengers along the ten miles of track from the Pen-y-Darren Ironworks to the Glamorganshire Canal. This historic event saw the world's first steam locomotive to run on rails hauling a train carrying fare-paying passengers.

In 1812 Oliver Evans, a United States engineer and inventor, published his vision of what steam railways could become, with cities and towns linked by a network of long distance railways plied by speedy locomotives, greatly reducing the time required for personal travel and for transport of goods. Evans specified that there should be separate sets of parallel tracks for trains going in different directions. Unfortunately, conditions in the infant United States did not enable his vision to take hold. However, this vision had its counterpart in Britain, where it proved to be far more influential. William James, a rich and influential surveyor and land agent, was inspired by the development of the steam locomotive to suggest

a national network of railways. He was responsible for proposing a number of projects that later came to fruition, and he is credited with carrying out a survey of the Liverpool and Manchester Railway. Unfortunately, he became bankrupt and his schemes were taken over by George Stephenson and others. However, he is credited by many historians with the title of "Father of the Railway".

Between 1814-21 Northumbrian engineer George Stephenson built 17 experimental locomotives. Although he was not the first to produce a steam locomotive, he was the prime mover in introducing them on a wide scale. His turning point came in 1821 when he was appointed engineer-in-charge of what became the 42 km (26 miles) long *Stockton and Darlington Railway*, running from the County Durham towns of Stockton-on-Tees, a seaport, and Darlington, an industrial center and was intended to enable local collieries to transport their coal to the docks. As coil would constitute the bulk of the traffic, the company took the important step of offering to haul the colliery wagons or chaldrons by locomotive power, something that required a scheduled or timetabled service of trains. It was opened in 1825 and Stephenson's Locomotion No.1 drew the first train. This historic event saw the world's first public railway regularly to use steam locomotives to haul wagons of goods (the main traffic was coal) and carriages of passengers. In fact, the line also functioned as a toll railway, where private horse drawn wagons could be operated upon it. Passengers were carried in horse-drawn coaches until 1833. This curious hybrid of a system (which also included, at one stage, a horse drawn passenger wagon) could not last, and within a few years, traffic was restricted to timetabled trains. (However, the tradition of private owned wagons continued on railways in Britain until the 1960s.)

The success of the Stockton and Darlington encouraged the rich investors of the rapidly industrialising North West of England to embark upon a project to link the rich cotton manufacturing town of Manchester with the thriving port of Liverpool. The *Liverpool and Manchester Railway* was the first modern railway, in that both the goods and passenger traffic was operated by scheduled or timetabled locomotive hauled trains. At the time of its construction, there was still a serious doubt that locomotives could maintain a regular service over the distance involved. In 1829, Lancashire's Liverpool and Manchester Railway, built mainly to carry cotton, offered a 500 pound prize to the winner of a competition for the best steam-locomotive design to work on the line. The trials were held at Rainhill, near Liverpool. Of three locomotives entered, George Stephenson's Rocket, gaily painted in yellow, black and white, won at a speed of 26 mph (42 kph). The promoters were mainly interested in goods traffic, but after the line opened on 15 September 1830, they found to their amazement that passenger traffic was just as remunerative. The success of the Liverpool and Manchester railway influenced the development of railways elsewhere in Britain and abroad. The company hosted many visiting deputations from other railway projects, and many railwaymen received their early training and experience upon this line.

It must be remembered that the Liverpool and Manchester line was still a short one, linking two towns within an English shire county. The world's first trunk line can be said to be the *Grand Junction Railway*, opening in 1837, and linking a mid point on the Liverpool and Manchester Railway with Birmingham, by way of Crewe, Stafford, and Wolverhampton.

From then on, railways started to spread all around the world and the development of new technologies improved the system in such a way that railways and in particular high-speed rail represent now, with the ongoing threat of global warming and energy shortages, the key to the future of transportation in many of the world's developed countries.

1.2 Motivation

With the intensification of train-traffic, particularly on the so called European corridors, the complexity of railway operations have increased significantly and large railway stations with hundreds of trains must be handled per day. The management of busy, complex railway infrastructures represents now a big challenge for transport systems and greater traffic densities together with an increasing competitive pressure among train operators require a better usage of the resources.

Nowadays we can benefit from the availability of effective, computer-aided tools to improve the management ability of railways over traditional methods, and this is consequent to the new market scenario. In many European countries, railways are being transformed into more liberalized and privatized companies, which are expected to compete on a more profit-oriented basis by improving the overall efficiency and quality of service of the railway transport system. Moreover, the strategic character of the sector is highlighted in view of ecological impacts and national policies aimed at spilling freight traffic shares from roads to rails.

The task of the *Infrastructure Manager* is related to *train planning* and *real-time control*, whereas different *Operators* are responsible for *rolling stock* and *train services*. Train planning means optimising the usage of railway infrastructure (railway *lines* or *corridors* and *stations* or *nodes*) according to operators requests for capacity on common railway lines and railway stations. In this context, the ability to undertake railway management in a timely and efficient way is a compelling need for the Infrastructure Manager, who has to solve several planning problems, following the normative directives introduced in the European Union since early 90s. For this reason, operations research and combinatorial optimisation models represent a thriving tool to solve railway problems. Thanks to mathematical planning models, the Infrastructure Manager can allocate "at best" the resources requested by all transport operators and proceed with the overall design process, possibly with final local refinements and minor adjustments, as in the tradition of railway planners.

Even though extensive research has been carried out in this field, most of it is related to the construction of good timetables for heavy traffic, long-distance corridors, i.e. optimisation for railway lines (called *Train Timetabling*). But what is missing is optimisation for the railway stations. In fact, corridors connect several different nodes, and it is the latter that often represent important bottle-necks for traffic management. *Train Platforming*, also known as train routing, is the follow-up to train timetabling and takes into account the actual routing (definition of arrival and departure routes) and platforming (definition of the stopping platform) of the trains within each single railway node. Stations with several hundreds of trains per day are common throughout all Europe and elsewhere. Such stations typically have multiple intersecting in-lines and out-lines, connected to multiple parallel or sequential stopping points (platforms), of different types and lengths, some being dead-end and some being one or two-ways through-platforms. The trains differ in their types, speeds, desired dwell times and headways, origins an destinations, and preferred or required lines and platform. For safety and other reasons, headways (time gaps) are required before and after each train arrival and departure at each platform and on each line. There may be other temporary or permanent constraints or preferences due to the station layout, safety or signalling requirements, operating policy or marketing policy. Train Platforming for a large, multi-platform, busy station with multiple in-line and out-lines means drawing up a plan to ensure that there are no conflicts between any trains, while ensuring that all the above requirements and constraints are satisfied and minimising any deviations, or costs of deviations, from desired or

preferred times, platforms or lines for each train.

The main, driving case-study of this work is Train Platforming, for which in Chapter 2 we present an exact optimisation algorithm that allows the Infrastructure Manager, once the timetable is given, to define routes and platforms for all trains at minimum cost. Extensive testing has been carried out on real-world instances from the Italian Infrastructure Manager (RFI).

The successful solution of train platforming via integer programming, somehow sets nowadays state-of-the-art, where optimisation methods have reached a state of maturity as a consequence of decades of research. Despite this success, we still face severe difficulties when we try to apply algorithms to real-world contexts, especially for railway problems. Often, our methods prove to be rather awkward when applied to a real setting, because they cannot cope at all or at least rather weakly with the idea of *imperfect information*. In fact, the complexity and size of such optimisation problems is boosted by the inevitable presence of disturbances that change input data and adversely affect the correctness and effectiveness of our plan. The idea of *Robust Optimisation* was introduced more than thirty years ago, but still it has proved to be rather immature when applied to real-world railway problems. Hence, this setting poses new challenges for robust mathematical optimisation models and requires a new paradigm for Robust Optimisation. For this reason a new concept has been recently developed within the EU ARRIVAL-project (2006-2009): *Recoverable Robustness*. The European scientific community was interested in advancing the current state of algorithmic research by considering perhaps the most complex and largest in scale problems: those of railway systems. In particular, the two main features of planning that have been attacked in this research project were: *robust* planning and *online* (*real-time*) planning. These two represent important and rather unexplored areas of the planning process, and they both contribute to make optimisation problems even harder. Recoverable Robustness is a new paradigm for Robust Optimisation that joins in a new mathematical framework the idea of robustness with the idea of recovery and represents a flexible answer to over-conservativeness of classical models. Chapter 3 is dedicated to the mathematical definition of Recoverable Robustness (for a complete treatment see Liebchen et al. [28] and Stiller [34]). More details are given for a specific method within the Recoverable Robustness framework, called *Network Buffering*. This technique relies on the simplest recovery policy: *delay propagation*. Even if simple, the method proved to be rather effective. In fact, in Chapter 5 we examine the very first application of Recoverable Robustness via Network Buffering to a real-world context: that of Train Platforming.

Together with the idea of Recoverable Robustness, also the *Price of Recoverability* (PoR) was defined as a measure of recoverability in [28]. The PoR measures both the objective function of the original problem and the recovery costs. This concept is presented in Chapter 3 and then it is practically applied in Chapter 4. In particular, we explore a way to calculate lower bounds for the PoR in a real-world application: *tactical Rolling Stock Scheduling* at NS (the main operator in the Netherlands).

Finally the last chapter, still work in progress, is dedicated to evaluating different Robust Optimisation models and *re-scheduling* policies in an *online (real-time)* setting via *simulation* framework. The purpose is to test different robust models and recovery algorithms and measure the effectiveness of each pair, by calculating the total delay propagated in the system.

In the next section, we will give a detailed overview on each chapter. For reading purposes, please note that Chapters 4-6 rely on the definitions of Chapter 2 and 3, but they are independent among themselves.

1.3 Contents: an overview

1.3.1 Train Platforming

Train platforming is basically concerned with finding an assignment of trains to platforms in a railway station, even though the complete version of the problem includes the definition of the trains routes within the infrastructure. Routing a train in a railway station means finding two paths that link the arrival and departure directions of the railway line for the given train to a stopping platform. Thus, choosing a platform means defining a subset of the available paths, those connecting the platform to the train directions. This problem is relatively easy for small contexts, i.e. stations with a few platforms and a few alternative paths to route the trains, but becomes an extremely difficult task when applied to complex topologies. The essential characteristics of the process can be summarized as follows:

- the station structure is described through a network model with platforms, directions and connecting paths; we are given the train timetable (with scheduled arrival and departure times, and tolerances within which they can be "changed"), priorities and preferences for platform allocation;
- the optimal allocation is found by minimising the overall cost of a penalty function, which takes into account the deviations of the assignments from their desired features, and the number of platforms used;

Although there has been abundant research on train timetabling, the platforming problem has not received a considerable attention so far. Most studies on railway management are not concerned with large, busy multi-platforms stations with multiple in-lines and out-lines or assume they have unlimited capacity. However, in Europe, busy train stations with complex topologies are very common and represent an important factor in rail network planning. De Luca Cardillo at al. in [17] considered a simplified version in which, for each train, the scheduled arrival and departure times cannot be changed and the paths used to route the train within the station are uniquely determined by the choice of the platform. In this case the assignment incompatibilities can be represented implicitly through a list of incompatible train-platform pairs of the form (j, a, k, b), stating that it is infeasible to assign train j to platform a as well as train k to platform b. It is important to notice that, in this case, the problem can be modelled as a graph coloring problem with some additional constraints, for which the authors propose a heuristic algorithm applied to real-world instances concerning stations with up to 13 platforms and 177 trains.

The same platforming version was addressed by Billionet in [6], who considered a classical integer linear programming formulation for graph coloring and showed how to incorporate into the model the clique constraints associated with the list of incompatible train-platform pairs. Randomly-generated instances with 200 trains and up to 14 platforms are solved using a general-purpose integer linear programming solver.

A more general version of the problem, in which arrival and departure times and arrival and departure routes are not fixed a priori, is addressed in Zwaneveld et al. [37] and [35]. This version is modelled as a Stable Set Problem on a graph in which each node corresponds to a choice for a train, with an associated cost, and edges join node pairs associated with the same train as well as node pairs corresponding to incompatible choices for different trains. The problem is naturally formulated as an integer linear program with one binary variable for each node and clique inequality constraints.

Finally the version addressed by Carey et al. in [13] is intermediate between the two illustrated so far. In fact, the arrival and departure times can be changed, but the assignment of a train to a platform uniquely determines the routes that the train will follow on his way to and from the platform. The paper discusses in great detail the conflicts that may arise from the assignments and the procedures that are followed to evaluate the costs. Rather than a mathematical formulation, the authors describe a constructive heuristic that is applied to a real-world instance from the British railways concerning the station of Leeds.

The train platforming problem we consider in this work is based on a rather general formulation proposed by the Italian Infrastructure Manager, where, for each train, we aim at defining the stopping platform, together with the arrival and departure paths. This version also allows modifications on the train schedule (within some range). This last feature is particularly important, since it represents a feedback among two consecutive and strictly interconnected planning phases: *timetabling* (optimisation of the railway lines) and *platforming* (optmisation of the railway stations). We illustrate an ILP formulation for the problem and an exact algorithm tested using real-world data. The results represent an important step forward in railway planning. For the instances in our case study, we show that we can produce solutions that are much better than those produced by a simple heuristic currently in use, and that often turn out to be (nearly-)optimal.

1.3.2 Recoverable Robustness

The idea of Robust Optmisation was introduced more than 30 years ago by Soyster [33] and it is a concept for optimisation under uncertainty. Optimisation under uncertainty seeks for good solutions although the input data, or part of the input data is not known exactly. Stochastic programming is concerned with the same type of setting, hence the two concepts are strictly related.

Stochastic programming methods, given a probability distribution for the input data, optimise an expected value or an objective including some risk measure. Stochastic programs usually have two stages. In the first stage, decisions are taken without knowledge of the data, in the second stage these decisions can be adjusted to the now known exact data.

Robust optimisation takes a different approach: a robust solution is a solution that is guaranteed to be feasible for every realisation of the input data (*scenarios*). This has two main advantages. Firstly, handling of distributions is not needed, and this saves consistent computational effort. Secondly, one can rely on a robust solution in every likely situation (the scenarios considered), not just on average.

This last feature *guarantees* the given solution against any input occurrence (considered), but it also represents the main drawback of classical robust optmisation methods. In fact, the strict constraint on the feasibility of a robust solution for *all* the scenarios that are considered makes the approach extremely over conservative. In robust programming there is no second stage decision: the first stage decision must be feasible as it is for all considered scenarios. In real-world applications, the scenario set can be extremely broad, thus the cost of a robust solution can get easily very large and unacceptable for the planner. In many cases, such a solution does not even exist. In fact, scenarios can be extremely different from each other and the solution spaces may not intersect. This rules out classical robust optimisation for many applications.

On the other hand, in most real-world problems, the level of robustness required is far less strict than the one formalised in the classical robust optimisation concept, because solutions

may be *recovered* after the data has changed. In many applications, limited and simple means of recovery are provided quite naturally, and excluding them from the model destroys the vital flexibility of the system. For example, a timetable without the flexibility of small delays for some events must be planned very loosely in advance even to cope with small disturbances. Hence, the common pitfall of all classical robust optimisation methods is that they never consider the possibility to adjust the nominal solution according to the realised scenario. Nevertheless, adjustability or recovery can almost always be applied in real-world situations and ignoring this aspect, sets a huge gap between mathematical approaches and their practical utilisation.

Moreover, strict robustness is unable to account for limits to the sum of all disturbances, whereas, in practice, it can often be inferred that the number of constraints affected by disturbances is limited. For example, only a few train rides experience seminal disturbances. From a strict robust perspective, hedging against those limited scenarios is equivalent to hedging against scenarios in which all train rides have seminal disturbances.

All in all, solutions are expected to be feasible for a limited set of scenarios and for a limited recovery in each scenario. To overcome this principal weakness of classical robustness, the concept of *Recoverable Robustness* has been introduced by [28]. *Classical robustness*, also called *strict robustness*, is a special case of this new concept. The notion of recoverable robustness has been developed within the EU ARRIVAL-project inter alia to overcome these weaknesses of strict robustness. Informally speaking, a solution to an optimisation problem is called recovery robust if it is likely that it can be adjusted to the actual scenario by limited changes. Thus, a recovery-robust solution also provides a service guarantee. More formally, a solution x is *Recoverable Robust* against certain scenarios and for a certain limited type of recovery, if for each of the considered scenarios the solution x can be recovered to a feasible solution within the limits of the type of recovery. Thus, recoverable robustness allows for second stage decisions, the so-called recovery. This provides for a broadening of modeling power that is desirable in many real-world applications.

1.3.3 Price of Recoverability for Railway Rolling Stock Planning

Recently, within the EU ARRIVAL-project, the concept of Recoverable Robustness as a generic framework for modelling robustness issues in railway scheduling problems has been introduced by [28]. Also, the *Price of Recoverability* (PoR) was defined as a measure of recoverability. However, this notion is mainly of theoretical nature, and cannot be used in a straightforward way for concrete problems. In particular, computing the PoR requires, according to [28], minimisation over a set of recovery algorithms, and it is not clear how this can be carried out. For this reason, we wanted to investigate another way of bringing the theoretical notion of PoR closer to practice. In particular, we consider what happens if recovery is done in the best possible way: the resulting value of PoR for this unique "optimal" recovery algorithm is then a lower bound on the value of PoR for *any* set of recovery algorithms. More specifically, we address the practical evaluation of the lower bound above for a specific case study, concerning the medium-term rolling stock planning problem of NS, the main operator of passenger trains in the Netherlands. It arises 2–6 months before the actual train operations, and amounts to assigning the available rolling stock to the trips in a given timetable. The objectives of the problem that is traditionally solved, called nominal problem, are related to service quality, efficiency, and – to a limited extent – to robustness. Reference [20] describes a Mixed Integer Linear Programming (MILP) model for this nominal problem. The solutions

of the nominal problem are optimal under undisrupted circumstances only. However, infrastructure failures, bad weather, and engine break-downs often lead to disruptions where the nominal solution cannot be carried out any more. In such cases, disruption management must take place to come up with an adjusted rolling stock schedule. In this re-scheduling process, the original objective criteria are of marginal importance, the goal being to quickly find a feasible solution that is "close" to the nominal one and can be implemented in practice.

The computation of the lower bound on the PoR mentioned above for our case study requires the solution of a very-large Linear Programming (LP) model, in which several possible disruption scenarios are considered. We propose a Benders decomposition approach for the solution of this LP, leading to a subproblem for each scenario. Our preliminary computational results on a real-life rolling stock planning instance of NS indicate that the method takes relatively short time, and widely outperforms the straightforward solution of the whole LP by a state-of-the-art solver.

1.3.4 Recovery-Robust Platforming by Network Buffering

We aim at assigning platforms, arrival and departure routes to trains at a station in such a way that the total delay in the system caused by some inevitable, small, seminal disturbances in every-day operations is guaranteed to stay below some minimised bound. To this end, we apply the concept of recoverable robustness, in particular *network buffering*, to the standard train platforming problem.

In many cases, as in our real-world study on two main stations of the Italian railway network, the platforming capacity of a station is a bottleneck of the whole system, so tight planning is required. As a consequence, even small disturbances to the schedule can widely affect the smooth operation of the system. Therefore, in this study we construct a platforming plan which can be operated with only a limited amount of total delay and without re-planning of tracks or paths in all likely scenarios, i.e., in those scenarios with a limited amount of small disturbances. In other words, we seek a *robust platforming*.

Robust optimisation finds best solutions, which are feasible for all likely scenarios. This has the advantage that no knowledge of the underlying distribution is required, taking into account that acquiring data about the distribution is not always possible in practice. Moreover, robust models are usually easier to solve, because the solver need not handle distributions, nor look explicitly at each scenario to assess the objective. This is of particular relevance for the optimisation of large-scale systems like train platforming. Finally, robust solutions are guaranteed to be feasible in all likely scenarios. In many applications this is more desirable than, e.g., a satisfying performance of the solution on average. Hence, in our case, robust platforming will provide for smooth operations in all cases except those that feature extraordinarily high disturbances.

On the other hand, platforming problems feature both causes for over-conservatism in classical robust models. Platforming naturally allows for a limited, simple recovery, i.e., a limited amount of propagated total delay, and it makes sense to assume that only a limited number of trains will reach the station initially delayed, or experience a seminal disturbance on the platform before departure. We define the set of likely scenarios in this spirit.

For instance, for all scenarios with moderate disturbances, a *recovery-robust platforming* solution is guaranteed to be adjustable by simple means and at an a priori limited cost, i.e., the total delay will not exceed a certain a priori fixed budget. Moreover, recoverable robustness can fulfill its guarantee at a lower cost in the nominal objective, because it can

take into account restricted scenario sets for uncertainty on the right-hand side.

An advantageous feature of the platforming problem is that disturbances only occur on the right hand side of the problem. This enables the use of *robust network buffering*, which yields tractable recovery-robust models. Roughly speaking "tractable" means that, out of the infinite (uncountable) set of scenarios, we can restrict attention to a finite and relatively small set in modelling the problem. This leads to a tractable model for recovery robust platforming through network buffering. In addition, network buffering can be combined with existing specialized algorithms for the deterministic platforming problem.

In this way we could construct recoverable robust platforming plans for the two stations we considered in our computational study on real-world data, namely Genova and Palermo. The results are striking. While nominal optimality is maintained, the propagated delay can be reduced by up to 49% percent. The network buffering approach allows to calculate robust train platforming plans for real-world instances. The buffering could be attached to an existing platforming software. For all time windows and both train stations we considered, the robust plans achieve nominal optimality, i.e., they assign the maximal number of trains that any platforming can assign in the instance. But they assign them in such a way, that the maximal propagated delay in any scenario, with less than the total seminal disturbance, is reduced. In some cases, the propagated delay is almost halved by the robust approach. Note, that the deterministic method to which we compare the network buffering approach is not completely unaware of the tightness of the solutions: in fact it punishes the total time of (small) path conflicts. The computational study shows that this perspective is not sufficient to choose the most delay resistant among the nominally optimal platforming plans. Thus, this method finds significantly more reliable solutions without extra costs.

1.3.5 Robust Train Platforming and Online Rescheduling

As said, Train Platforming is a problem that arises in the early phase of the passenger railway planning process, usually several months before operating the trains. The main goal of platforming is to assign each train a stopping platform and the corresponding arrival/departure routes through a railway node. However, train delays often disrupt the routing schedules, thereby railway nodes are responsible for a large part of the delay propagation. Here we propose algorithms for computing robust routing schedules, and design a simulation framework to evaluate their robustness.

The railway stations of many European cities have a large throughput, so the routing task often turns out to be particularly challenging. Moreover, real-life train operations have to face delays that prevent the routing plan from being carried out, in which case the routing plan needs recovery measures to resolve the conflicts. Such measures include assigning new resources to trains and modifying their arrival and departure times (which may result in additional train delays). Yet, most papers generally focus on routing as a pure planning problem and there is little literature available on robustness and rescheduling (recovery) issues within this context.

Our goal is threefold. The first goal is to extend the existing platforming model of Caprara et al. [11] to robustness considerations, describing some extensions where robustness is enhanced either by increasing the delay absorption capacity or by explicitly providing potential recovery possibilities. The second goal is to design recovery algorithms for platforming and routing, possibly based on these extended models. The third goal is to assess the performance of these recovery algorithms in a simulation framework. Specifically, given a timetable of an

entire day, a routing plan and some randomly-generated delays, the recovery algorithms are applied to resolve the routing conflicts. Then, the simulation framework allows one to compare the robustness of different platforming and routing plans using certain recovery algorithms, the main criterion in the comparison being the sum of all train delays.

Robust Planning

We present different approaches to embed robustness in the routing plan, giving rise to three variants of the model of [11].

Basic In the first variant we simplify the original model, by considering exclusively the cost of the patterns, without any additional fixed cost for the platforms used. In fact, in a robust model it may be desirable for the trains to spread platform occupation among different resources. Even penalties for path conflicts are ignored.

Increasing the Absorption Capacity In the second variant, the robustness is captured by penalizing the cases in which an infrastructure element (a platform or a route) is utilized by two trains in a short succession. These may give rise to resource conflicts in case of train delays and thereby may lead to propagation of such delays. So the aim is to spread the load on the infrastructure in time and space. A routing plan optimised for this criterion is expected to cope with small, accumulating delays, without substantial recovery actions.

Back-up Platforms In the third variant, one assigns each train a "primary platform" as well as a "back-up platform". The primary platform is announced to the passengers and has to be used unless delays occur. Delayed trains may be diverted to their back-up platform allowing re-scheduling possibilities.

Recovery Algorithms

Once a (robust) routing plan is given, one has to test its effectiveness against delays by applying to it a recovery strategy, which may be a general one or it may be designed starting from the specific robust plan. We deal with three different recovery strategies.

Delay Propagation This is a general strategy in which each train keeps its originally-assigned route and the order of trains on the trajectories is not modified either. Thus conflicts are resolved by adjusting the arrival and departure times and by eventually propagating the delay.

Using Back-up Platforms This strategy is applicable when back-up platforms are available, and assign trains either to their planned platform or to their back-up platform, whichever is available earlier.

Full Recovery This recovery rule allows all possible changes on the original plan. Hence, it is the most powerful, but also the most time-consuming.

Simulation Framework

The simulation framework is used to assess the level of robustness of a routing plan and the performance of a recovery algorithm under realistic train delays. The input of the simulation framework consists in a routing plan, a delay distribution and a recovery algorithm. The simulation involves all train arrivals and departures at a single railway node on a given day. In-bound trains are subject to random delays according to some realistic probability distribution.

The recovery process takes place at every minute and works over the subset of trains arriving in the next hour time span. Each time the recovery process has to decide on assigning new resources to trains and on modifying their arrival and departure times. This simulation technique is referred to as *rolling horizon*, since the re-optimisation concerns a 1-hour horizon which rolls forward minute after minute.

Chapter 2

Train Platforming Problem

We present a general formulation of the train platforming problem, which contains as special cases all the versions previously considered in the literature as well as a case study from Rete Ferroviaria Italiana, the Italian Infrastructure Manager. In particular, motivated by our case study, we consider a general quadratic objective function, and propose a new way to linearise it by using a small number of new variables along with a set of constraints that can be separated efficiently by solving an appropriate linear program. The resulting integer linear programming formulation has a continuous relaxation that leads to strong bounds on the optimal value. For the instances in our case study, the solution approach based on this relaxation produces solutions that are much better than those produced by a simple heuristic method currently in use, and that often turn out to be (nearly-)optimal.

2.1 Introduction

The objective of train platforming, which is the routing problem that generally follows any timetabling phase, is to find an assignment of trains to platforms in a railway station. The practical relevance of the problem inspired the definition of a few different versions, which are relatively easy to solve for small contexts, corresponding to stations with very few platforms and alternative paths to route the trains, but become extremely difficult when applied to complex railway station topologies such as those associated with the main European stations, leading to instances with hundreds of trains and tens of platforms. Moreover, most versions are not concerned with the station topology and ignore the routing phase, whereas the main European stations frequently have complex topologies and the routing issue can be quite a complicated task.

In Figure 2.1 we show the topology of the Bari Central Station (BA C.LE), a large-size station in Italy. A main station typically has several external lines (generally with two tracks) connecting it to other main stations; these lines are called *directions* in our context. In Figure 2.1 these directions are named D1,. . . ,D5. Among these, e.g., direction D1 is associated with two tracks and direction D2 with one track only. Direction D5 is for trains going to/coming from a shunting area (or locomotive depot), called "D.L." in the picture.

Moreover, there are several points at which a train may stop within the station to download/upload passengers and/or goods; these points are called *platforms* in our context, and can be of different type and length, some being dead-end and some being through-platforms. The station in Figure 2.1 has 14 platforms overall, numbered in Roman in the picture. The 10

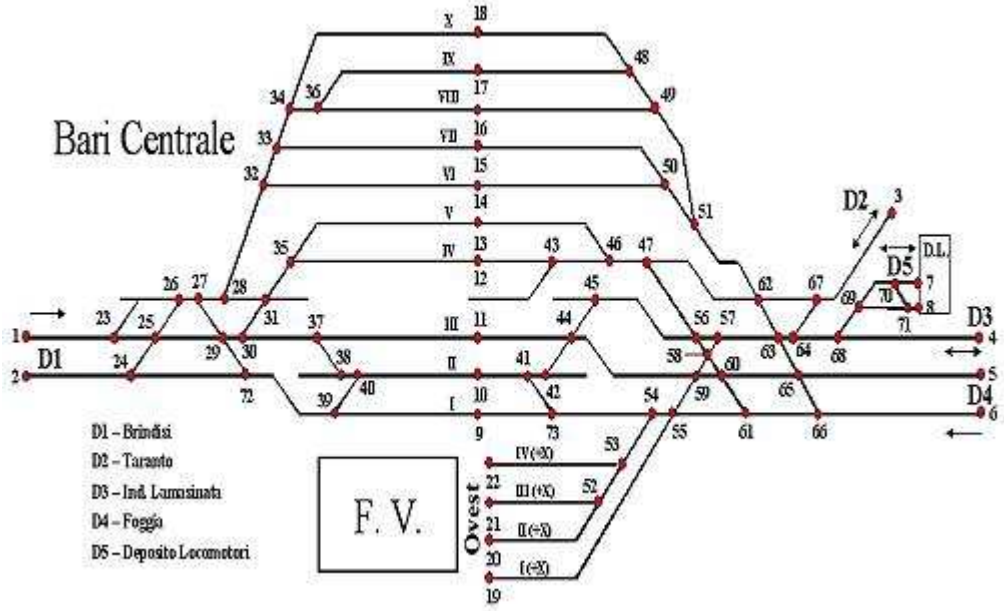

Figure 2.1: The layout of Bari Central Station (BA C.LE).

platforms numbered I,. . . ,X are "through" platforms, in which generally the trains arrive and depart travelling in the same direction, whereas the 4 platforms numbered I+X,. . . ,IV+X are "terminal" platforms, in which trains have to depart in the opposite direction with respect to the one in which they arrived. "F.V." indicates the passenger access to the station.

The connection between directions and platforms is achieved by internal lines, called *paths* in our context, which define a route within the station linking a given direction to a given platform. *Arrival paths* can be used to go from an arrival direction to a platform, *departure paths* can be used to go from a platform to a departure direction, and *two-way paths* can be used for both purposes. For instance, for the station in Figure 2.1, the path from direction D1 to platform I passing through the nodes numbered 1,23,25,29,72,39,9 is an arrival path, the path from platform I to direction D1 passing through the nodes numbered 9,39,72,24,2 is a departure path, and the path from/to direction D2 to/from platform I passing through the nodes numbered 3,67,64,63,57,58,59,55,54,73,9 is a two-way path.

The problem aims at defining for each train the platform where it will stop and the corresponding arrival and departure paths, while ensuring that all the constraints are satisfied and minimising a suitable objective function taking into account the platforms that are actually used, the deviation between actual and "desired" stopping platforms, and possible conflicts on the arrival and departure paths (in some cases, small deviations from the scheduled arrival/departure times are also allowed and considered in the objective function). Moreover, depending on the particular context, there may be other constraints due to the particular station layout, safety or signalling requirements, operating or marketing policies.

In this chapter, we propose a general formulation of the problem, along with an *Integer Linear Programming* (ILP) formulation whose *Linear Programming* (LP) relaxation is used to drive a heuristic algorithm that turns out to widely outperform a simple heuristic method currently in use for the instances in our case study. We consider a general quadratic objective function, given that the objective function is indeed quadratic in our case study, and propose

an efficient way to linearise it by using a small number of new variables along with a set of constraints that can be separated efficiently by solving an appropriate LP.

2.1.1 Literature review

In the following, we try to give a very quick but comprehensive view of the existing work, referring to the survey by Caprara et al. [12] for a more detailed description. As it is often the case with this type of problems, every reference generally considers a different version, making it difficult to compare the proposed methods. The easiest version is the one considered by De Luca Cardillo and Mione [17] and Billionet [6], who address a simplified version in which, for each train, the scheduled arrival and departure times cannot be changed and the paths used to route the trains within the station are uniquely determined by the choice of the platform. A more general version of the problem, in which arrival and departure times and arrival and departure paths are not fixed a priori is addressed in Zwaneveld [36], Zwaneveld et al. [35], Zwaneveld et al. [37], Kroon et al. [27]. Finally, the version addressed in Carey and Carville [13] is an intermediate one, in that arrival and departure times can be changed, but the assignment of a train to a platform uniquely determines the paths that the train will follow on its way to and from the platform.

2.1.2 The general problem considered

In this chapter, we deal with a general version of the problem, for convenience referred to in the sequel as the *Train Platforming Problem* (TPP). The specific versions previously considered in the literature, as well as the version of our case study, are special cases of TPP.

In this general version, we are given a set B of platforms and a set T of trains to be routed to a platform every day of a given time horizon. Moreover, for each train $t \in T$, we are given a collection $\mathcal{P}_t$ of possible *patterns*. Each pattern corresponds to a feasible route of train t within the station, including a stopping platform, an arrival path and an arrival time, a departure path and a departure time. Each train must be assigned a pattern that will be repeated every day of the time horizon (see Section 2.1.3 for examples of patterns associated with the station depicted in Figure 2.1).

Operational constraints forbid the assignment of patterns to trains if this implies occupying the same platform at the same time, or also using arrival/departure paths that intersect at the same time or too close in time. In the general version, this is represented by defining a pattern-incompatibility graph with one node for each train-pattern pair (t, P), with $P \in \mathcal{P}_t$, and an edge joining each pair $(t_1, P_1), (t_2, P_2)$ of incompatible patterns (see the graph depicted in Figure 2.2: the red nodes correspond to the patterns of train t_1 and the blue nodes to train t_2, an edge joins pairs of patterns that are incompatible). This graph models "hard" incompatibilities that must be forbidden in a feasible solution. However, in the general version, there are also "soft" incompatibilities, generally associated with the use of arrival/departure paths close in time, that are admitted but penalised in the objective function, as illustrated below.

TPP requires the assignment of a pattern $P \in \mathcal{P}_t$ to each train $t \in T$ so that no two incompatible patterns are assigned and the *quadratic* objective function defined by the following coefficients is minimised. There are a cost c_b for each platform $b \in B$ that is used in the solution, a cost $c_{t,P}$ associated with the assignment of pattern $P \in \mathcal{P}_t$ to train $t \in T$, and a cost c_{t_1,P_1,t_2,P_2} associated with the assignment of pattern $P_1 \in \mathcal{P}_{t_1}$ to train t_1 and the

assignment of pattern $P_2 \in \mathcal{P}_{t_2}$ to train t_2 for $(t_1, t_2) \in T^2$, in case these two patterns have a "soft" incompatibility. Here, $T^2 \subseteq \{(t_1, t_2) : t_1, t_2 \in T, t_1 \neq t_2\}$ denotes the set of pairs of distinct trains whose patterns may have a "hard" or "soft" incompatibility.

A key issue of our approach is to avoid, in the model formulation, the canonical approaches to linearise the objective function, e.g., by introducing additional binary variables to represent the product of the original binary variables; the number of these variables would be very large and the resulting LP relaxation fairly weak. This will be illustrated in detail in Section 2.2.2.

For the applications we are aware of, including our case study, the overall number of patterns $\sum_{t \in T} |\mathcal{P}_t|$ allows one to handle explicitly all of them. The model that we will present is valid even if this is not the case. As to the solution approach, we will illustrate it assuming the explicit list of patterns is given. If this is not the case, the applicability of the method strongly depends on the specific way in which patterns are described implicitly.

2.1.3 Our case study

The instances in our benchmark come from Rete Ferroviaria Italiana (RFI), the Italian Infrastructure Manager. The resulting problem is the special case of TPP with the following characteristics.

The set B of platforms includes *regular platforms*, corresponding to platforms that one foresees to use, and *dummy platforms*, corresponding to virtual additional platforms that one would like not to use but that may be necessary to find a feasible solution. Although the objective function contains many terms, the primary objective is to use as few dummy platforms as possible. In the example of Figure 2.1, the 14 regular platforms are those reported in the figure, whereas the three dummy platforms that guarantee the existence of a solution for each value of the parameters are not depicted.

Besides sets T and B, we also have a set D of *directions* for train arrivals and departures and a collection $\mathcal{R}$ of *paths* connecting directions and platforms. Some of these directions are associated with *shunting areas* for the trains that begin/end at the station. For each direction $d \in D$, we have a *travel time* g_d for all paths connecting d to any platform (independent of the specific path, platform, and train). Moreover, for each ordered pair $(d_1, d_2) \in D \times D$ corresponding to arrival direction d_1 and departure direction d_2, the input specifies a *preference list* $L_{d_1,d_2} \subseteq B$ of preferred platforms for all trains that arrive from direction d_1 and depart to direction d_2. In the example of Figure 2.1, $D = \{$D1,D2,D3,D4,D5$\}$. Collection R contains 89 paths, among which 24 are arrival paths, 24 are departure paths, and 41 are two-way paths.

For each direction $d \in D$ and platform $b \in B$, we have a (possibly empty) set $\mathcal{R}_{d,b} \subseteq \mathcal{R}$ of paths linking direction d to platform b. Specifically, we have $\mathcal{R}_{d,b} = \mathcal{R}^{\mathrm{a}}_{d,b} \cup \mathcal{R}^{\mathrm{d}}_{d,b}$, where the paths in $\mathcal{R}^{\mathrm{a}}_{d,b}$ are *arrival paths* to get from d to b and $\mathcal{R}^{\mathrm{d}}_{d,b}$ are *departure paths* to get from b to d. Note that we may have paths in $\mathcal{R}^{\mathrm{a}}_{d,b} \cap \mathcal{R}^{\mathrm{d}}_{d,b}$, called *two-way paths*. For each path $R \in \mathcal{R}$, we are given a list $\mathcal{I}_R \subseteq \mathcal{R}$ of *incompatible* paths. (In particular, a path R is always incompatible with itself.) In the example of Figure 2.1, the already-mentioned arrival path (passing through nodes) 1,23,25,29,72,39,9 and departure path 9,39,72,24,2 are incompatible since they intersect out of platform I, whereas both paths are compatible with the two-way path 3,67,64,63,57,58,59,55,54,73,9 since their unique intersection with this path is at platform I.

Each train $t \in T$ has an associated *ideal arrival time* u^{a}_t at a platform, along with a maximum *arrival shift* s^{a}_t, and an associated *ideal departure time* u^{d}_t from the platform, along

with a maximum *departure shift* s_t^{d}, meaning that the train must arrive to a platform in the interval $[u_t^{\mathrm{a}} - s_t^{\mathrm{a}}, u_t^{\mathrm{a}} + s_t^{\mathrm{a}}]$ and depart in the interval $[u_t^{\mathrm{d}} - s_t^{\mathrm{d}}, u_t^{\mathrm{d}} + s_t^{\mathrm{d}}]$. Moreover, each $t \in T$ has an associated arrival direction $d_t^{\mathrm{a}} \in D$, a departure direction $d_t^{\mathrm{d}} \in D$ and a set $C_t \subseteq B$ of candidate platforms where it may stop, corresponding to the platforms for which there exist at least two paths linking respectively the arrival and departure directions of t to the given platform. I.e., $C_t = \{b \in B : \mathcal{R}^{\mathrm{a}}_{d_t^{\mathrm{a}},b} \neq \emptyset, \mathcal{R}^{\mathrm{d}}_{d_t^{\mathrm{d}},b} \neq \emptyset\}$. In our case study we have $s_t^{\mathrm{a}} = s_t^{\mathrm{d}} = 1$, i.e. the maximum arrival and departure shifts are one minute. In the example of Figure 2.1, there are 237 trains on input, among which the regional train t with code 22272 has arrival direction D1 with ideal arrival time 7:01 and departure direction D4 with ideal departure time 7:04. This implies that, in the solution the train must arrive in the interval [7:00,7:02] and depart in the interval [7:03,7:05]. In this case, we have $C_t = \{\mathrm{I},\ldots,\mathrm{X}\}$, i.e., train t can stop in all ten "through" platforms.

A pattern $P \in \mathcal{P}_t$ is defined by a platform $b \in C_t$, an arrival path $R^{\mathrm{a}} \in \mathcal{R}^{\mathrm{a}}_{d_t^{\mathrm{a}},b}$, a departure path $R^{\mathrm{d}} \in \mathcal{R}^{\mathrm{d}}_{d_t^{\mathrm{d}},b}$, and the corresponding *actual arrival time* $v_t^{\mathrm{a}} \in [u_t^{\mathrm{a}} - s_t^{\mathrm{a}}, u_t^{\mathrm{a}} + s_t^{\mathrm{a}}]$ and *actual departure time* $v_t^{\mathrm{d}} \in [u_t^{\mathrm{d}} - s_t^{\mathrm{d}}, u_t^{\mathrm{d}} + s_t^{\mathrm{d}}]$. Conventionally, the pattern occupies platform b for the interval $[v_t^{\mathrm{a}} - h, v_t^{\mathrm{d}} + h]$, where h is a so-called *headway* value introduced for safety reasons. Moreover, the pattern occupies arrival path R^{a} for the interval $[v_t^{\mathrm{a}} - g_{d_t^{\mathrm{a}}}, v_t^{\mathrm{a}}]$ and the departure path R^{d} for the interval $[v_t^{\mathrm{d}}, v_t^{\mathrm{d}} + g_{d_t^{\mathrm{d}}}]$, recalling the travel times defined above. In our case study, we have $h = 3$. Moreover, for the example of Figure 2.1 we have $g_{d_t^{\mathrm{a}}} = g_{d_t^{\mathrm{d}}} = 4$ for all $d \in D$. Considering the train t with code 22272 mentioned above, a possible pattern is associated with platform I, arrival path 1,23,25,29,72,39,9 and arrival time 7:01, departure path 9,73,54,55,59,60,65,5 and departure time 7:03. This pattern occupies platform I for the interval [6:58,7:06], its arrival path for the interval [6:57,7:01], and its departure path for the interval [7:03,7:07].

The actual arrival and departure times must be in (an integer number of) minutes, which strongly limits the total number of patterns. Moreover, the problem is periodic with period 1440 minutes (one day), and therefore all times should be considered modulo this period.

Two patterns $P_1 \in \mathcal{P}_{t_1}$ and $P_2 \in \mathcal{P}_{t_2}$ are incompatible if either their platform occupation intervals overlap for a time window of duration greater than 0, or if they occupy incompatible paths for a time window of duration greater than π, where π is a so-called *dynamic threshold*. Note that there may be two disjoint time windows in which P_1 and P_2 occupy incompatible paths (e.g., one time window associated with incompatible arrival paths and one associated with incompatible departure paths), and in this case P_1 and P_2 are incompatible if and only if the largest duration between the two time windows is greater than π. For instance, in case $\pi = 2$, the above-mentioned pattern for the train with code 22272 is incompatible with the pattern associated with platform X, arrival path 1,23,25,26,27,28,32,33,34,18 and arrival time 6:59 (i.e., occupying its arrival path for the interval [6:55,6:59]), departure path 18,48,49,51,62,63,65,5 and departure time 7:04 (i.e., occupying its departure path for the interval [7:04,7:08]): indeed, the two arrival paths are incompatible and their occupation time windows overlap for 2 minutes, and the two departure paths are incompatible and their occupation time windows overlap for 3 minutes, i.e., the maximum overlap is $3 > 2$. If $\pi = 3$, the two patterns are not incompatible although a cost is incurred if both are selected in a solution (see below).

The fact that dummy platforms are virtual is modelled by associating with each dummy platform b infinitely many two-way paths for each direction $d \in D$, all of which are compatible with each other, meaning that the only incompatibilities between trains stopping at b are

related with the occupation of platform b itself (still associated with headway h), as the trains can always use compatible arrival and departure paths.

The objective function is computed by using the following coefficients, for which we also report the numerical values to give an idea of their relative importance: $\alpha_1 = 1000$, $\alpha_2 = 100000$, $\alpha_3 = 1$, $\alpha_4 = 100$, $\alpha_5 = 10000$, $\alpha_6 = 5$.

Each platform cost is given by $c_b = \alpha_1$ if b is a regular platform, and $c_b = \alpha_2$ if b is a dummy platform (in other words the cost for using a dummy platform is two orders of magnitude larger than the cost for using a regular platform).

Each coefficient $c_{t,P}$ is given by $\alpha_3 \cdot p_t \cdot s_P$, where p_t is a *train priority* value given on input and s_P is the total shift of pattern P (counting both the arrival and departure shifts), plus α_4 if pattern P stops at a regular platform not in the preference list $L_{d_t^{\mathrm{a}},d_t^{\mathrm{d}}}$, plus α_5 if, instead, the pattern stops at a dummy platform. In our case study, we have $p_t \in \{1, 2\}$ for $t \in T$.

Finally, each coefficient c_{t_1,P_1,t_2,P_2} is given by $\alpha_6 \cdot p_{t_1} \cdot p_{t_2} \cdot o_{P_1,P_2}$, where p_t is again the train priority and o_{P_1,P_2} is the sum of the durations of the (up to two, see above) time windows in which P_1 and P_2 occupy incompatible paths. For the two patterns mentioned above, say P_1 and P_2, which are not incompatible in case $\pi = 3$, we have $o_{P_1,P_2} = 2 + 3 = 5$.

2.1.4 Problem complexity

In this section we briefly discuss the fact that TTP is (strongly) NP-hard even if we consider two very restricted special cases. In both cases, there are no incompatibilities between arrival/departure paths, i.e., the only limited resources are the platforms. The first hardness result is based on the fact that the problem is periodic (with period one day).

Proposition 1. *TTP is strongly NP-hard even if (i) every train can go to every platform, (ii) two patterns are incompatible if and only if they occupy the same platform at the same time, and (iii) the objective is the minimisation of the number of platforms used.*

Proof. In case the arrival and departure times of the trains cannot be changed, representing the period of one day as a circle, each train corresponds to a circular interval of this circle, whose endpoints are its arrival and departure times. The problem is then to assign a platform (color) to each train (circular interval) so that overlapping intervals are assigned distinct colors and the number of used colors is minimised. This is exactly the Circular Arc Graph Coloring problem shown to be NP-hard in [24]. □

In case there is a point in time in which no train is on a platform (e.g., midnight), the Circular Arc Graph Coloring problem of the previous proof becomes an Interval Graph Coloring problem, which is easily seen (and well known) to be solvable efficiently. However, in case for each train only a subset of the platforms are feasible, the problem is again difficult.

Proposition 2. *TTP is NP-hard even if (i) there is a point in time in which no train is on a platform, (ii) two patterns are incompatible if and only if they occupy the same platform at the same time, and (iii) the objective is the minimisation of the number of platforms used.*

Proof. The circular period can be represented as a linear segment whose endpoints correspond to the point in time in (i), and each train corresponds to an interval of this segment. Moreover, in the feasibility version of the problem we have a set B of platforms, say $B = \{1, \ldots, n_b\}$,

and, for each train $t \in T$, we have a set $C_t \subseteq B$, and would like to assign a platform (color) $b \in C_t$ to each train (interval) $t \in T$ so that overlapping intervals are assigned distinct colors.

It is known that this problem (in recognition version) is strongly NP-complete even if $C_t = \{1, \ldots, b_t\}$ for $t \in T$, i.e., each train t must be assigned to one of the first b_t platforms, where b_t is given on input. Indeed, this is a possible equivalent reformulation of the following problem: given an interval graph $I = (N, E)$ (in which nodes in N correspond to intervals on the real line and an edge in E connects two nodes if and only if the corresponding intervals overlap) and a function $\mu : N \to \{1, \ldots, q\}$, decide if it is possible to color all the nodes of I with colors $\{1, \ldots, q\}$ so that adjacent nodes receive distinct colors and each node $j \in V$ receives a color at most equal to $\mu(j)$. This problem, called Interval Graph μ-Coloring in [8], has been shown to be strongly NP-complete in that paper. □

2.2 An ILP Formulation

In this section we present an ILP model for the general version of TTP that we consider. Recall the pattern-incompatibility graph G defined in Section 2.1.2, and let $\mathcal{K}$ be the whole collection of cliques in G (see an example of incompatibility clique depicted in Figure 2.3) and $\mathcal{K}_b$ be the collection of cliques in G associated with sets of patterns that use track b at the same time, i.e., for each $(t_1, P_1), (t_2, P_2) \in \mathcal{K}_b$, pattern P_1 for train t_1 and pattern P_2 for train t_2 both use platform b, occupying it for overlapping intervals.

The straightforward 0-1 quadratic programming formulation of the problem, using a binary variable y_b for each $b \in B$, with $y_b = 1$ if and only if platform b is used, and a binary variable $x_{t,P}$ for each $t \in T$ and $P \in \mathcal{P}_t$, with $x_{t,P} = 1$ if and only if train t is assigned pattern P, is the following:

$$\min \sum_{b \in B} c_b y_b + \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{(t_1,t_2) \in T^2} \sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ x_{t_1,P_1}\ x_{t_2,P_2} \tag{2.1}$$

subject to

$$\sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{2.2}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq y_b, \qquad K \in \mathcal{K}_b, \tag{2.3}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \tag{2.4}$$

$$y_b, x_{t,P} \in \{0, 1\}, \qquad b \in B,\ t \in T,\ P \in \mathcal{P}_t. \tag{2.5}$$

Constraints (2.2) guarantee that each train is assigned a pattern, constraints (2.3) impose that at most one train at a time occupies a given platform b, and if this ever happens that variable y_b takes the value 1, and constraints (2.4) forbid the assignment of patterns that are pairwise incompatible.

2.2.1 A convenient version of the clique inequalities

We first discuss how to modify constraints (2.3) and (2.4), whose number is exponential in the number of patterns, so that they can be handled in practice. First of all, each clique in $\mathcal{K}_b$

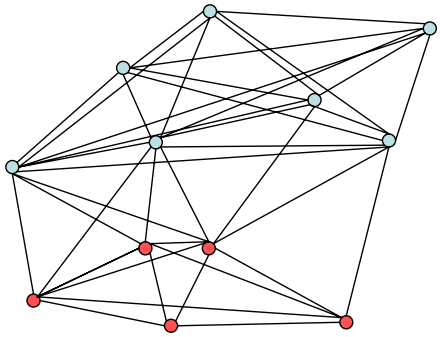

Figure 2.2: Incompatibility graph $G(t_1, t_2)$.

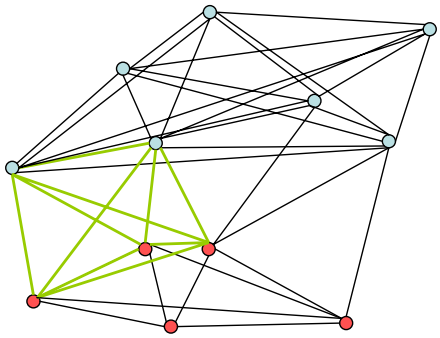

Figure 2.3: Clique of incompatible patterns in $G(t_1, t_2)$.

corresponds to a set of intervals (associated with the platform occupation of each pattern in the clique) that intersect pairwise. It is well known from the basic theory of interval graphs (see, e.g., [22]) that each maximal clique in such a graph is defined by an interval starting at point j together with all the intervals $[l, k]$ with $l \leq j$ and $k > j$. Therefore, the number of maximal cliques cannot be larger than the number of intervals. In our case, letting J_b denote the set of instants associated with the beginning of the occupation of platform b by a pattern, and $K(b, j) \subseteq \mathcal{K}$ the set of patterns that occupy platform b for an interval $[l, k]$ with $l \leq j$ and $k > j$, we have the following alternative version of constraints (2.3):

$$\sum_{(t,P)\in K(b,j)} x_{t,P} \leq y_b, \qquad b \in B,\ j \in J_b, \tag{2.6}$$

whose number, for our case study, is $\sum_{b\in B} |J_b| \leq |B| \sum_{t\in T}(2s_t^{\mathrm{a}} + 1)$, recalling that there are $2s_t^{\mathrm{a}} + 1$ possible instants at which train t may begin to occupy a platform.

As to constraints (2.4), they are in general hard to separate. However, if we restrict attention to cliques in $\mathcal{K}$ containing patterns of two trains only, we get a family of relaxed constraints that are still strong enough to be useful in practice (besides sufficing to define a model) and can be separated efficiently (provided the explicit list of all patterns is known), as explained in the next section. Given two trains t_1 and t_2, we let $\mathcal{K}(t_1, t_2) \subseteq \mathcal{K}$ denote the collection of cliques containing only patterns in $\mathcal{P}_{t_1} \cup \mathcal{P}_{t_2}$ and define the following alternative

version of constraints (2.4):

$$\sum_{(t_1,P_1)\in K} x_{t_1,P_1} + \sum_{(t_2,P_2)\in K} x_{t_2,P_2} \leq 1, \qquad (t_1,t_2)\in T^2, K \in \mathcal{K}(t_1,t_2). \tag{2.7}$$

In the next section, we discuss how to deal with this alternative version.

2.2.2 linearising the objective function

We finally illustrate how we linearise the quadratic term in the objective function (2.1). The textbook approach to linearisation amounts to introducing additional binary variables z_{t_1,P_1,t_2,P_2} that are forced to be one if $x_{t_1,P_1} = x_{t_2,P_2} = 1$ by imposing the linear constraints

$$z_{t_1,P_1,t_2,P_2} \geq x_{t_1,P_1} + x_{t_2,P_2} - 1. \tag{2.8}$$

The number of z variables is in this case very large and the resulting LP relaxation fairly weak. On the other hand, the following linearisation method requires a much smaller number of variables and leads to a stronger linear programming relaxation. We introduce the $\binom{|T|}{2}$ additional continuous variables w_{t_1,t_2} for $(t_1,t_2) \in T^2$, each representing, in the objective function, the term $\sum_{P_1\in\mathcal{P}_{t_1}} \sum_{P_2\in\mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ x_{t_1,P_1}\ x_{t_2,P_2}$. This leads to the linear objective function:

$$\min \sum_{b\in B} c_b\ y_b + \sum_{t\in T} \sum_{P\in\mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{(t_1,t_2)\in T^2} w_{t_1,t_2}. \tag{2.9}$$

We now show how to link the new w variables with the old ones, by first discussing how to do it in general and then by illustrating it through an example, to which the reader may refer while reading the general description.

An elementary link between the x and the w variables could be expressed by the linear constraints:

$$w_{t_1,t_2} \geq c_{t_1,P_1,t_2,P_2}\ (x_{t_1,P_1} + x_{t_2,P_2} - 1), \qquad (t_1,t_2)\in T^2,\ P_1\in\mathcal{P}_{t_1},\ P_2\in\mathcal{P}_{t_2}, \tag{2.10}$$

which would however lead to a model equivalent to the textbook one with the z variables and the constraints (2.8) mentioned above. Instead, we can define the following stronger inequalities to bound the w variables from below. Taking into account the assignment constraints (2.2) and observing that there are at most $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ possible values for w_{t_1,t_2}, we consider the simple polyhedron in $\mathbb{R}^{|\mathcal{P}_{t_1}|+|\mathcal{P}_{t_2}|+1}$ corresponding to the convex hull of the $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ possible values taken at the same time by vectors $(x_{t_1,P})_{P\in\mathcal{P}_{t_1}}$, $(x_{t_2,P})_{P\in\mathcal{P}_{t_2}}$ and by variable w_{t_1,t_2} in a solution. In particular, exactly one component of $(x_{t_1,P})_{P\in\mathcal{P}_{t_1}}$ and exactly one component of $(x_{t_2,P})_{P\in\mathcal{P}_{t_2}}$ take the value 1 in any feasible solution. In other words, there exist $P_1 \in \mathcal{P}_{t_1}$ and $P_2 \in \mathcal{P}_{t_2}$ such that $(x_{t_1,P})_{P\in\mathcal{P}_{t_1}} = e_{P_1}$ and $(x_{t_2,P})_{P\in\mathcal{P}_{t_2}} = e_{P_2}$, where, with a slight abuse of notation, for $i = 1,2$ we let e_{P_i} denote the binary vector in $\mathbb{R}^{|\mathcal{P}_i|}$ with the P_i-th component equal to 1 and all other components equal to 0. The corresponding value of w_{t_1,t_2} is c_{t_1,P_1,t_2,P_2}. Accordingly, the polyhedron that we consider is:

$$Q_{t_1,t_2} := \mathrm{conv}\{(e_{P_1}, e_{P_2}, c_{t_1,P_1,t_2,P_2}) : P_1\in\mathcal{P}_{t_1},\ P_2\in\mathcal{P}_{t_2}\}. \tag{2.11}$$

We can bound w_{t_1,t_2} in a way that is stronger than (2.10) by using valid inequalities for Q_{t_1,t_2} of the form:

$$w_{t_1,t_2} \geq \sum_{P_1\in\mathcal{P}_{t_1}} \alpha_{P_1}\ x_{t_1,P_1} + \sum_{P_2\in\mathcal{P}_{t_2}} \beta_{P_2}\ x_{t_2,P_2} - \gamma. \tag{2.12}$$

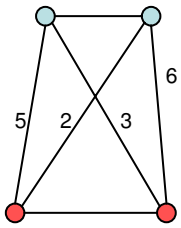

Figure 2.4: Weighted conflict graph of Example 1.

We let $\mathcal{F}_{t_1,t_2} \subseteq \mathbb{R}^{|\mathcal{P}_1|+|\mathcal{P}_{t_2}|+1}$ be the collection of vectors (α, β, γ) such that inequality (2.12) is valid for Q_{t_1,t_2} and not dominated by other valid inequalities. In the next section, we discuss how to deal with this family of inequalities.

Example 1. *Consider the very simple case in which* $\mathcal{P}_{t_1} = \{P_1, P_3\}$, $\mathcal{P}_{t_2} = \{P_2, P_4\}$, $c_{t_1,P_1,t_2,P_2} = 5$, $c_{t_1,P_1,t_2,P_4} = 3$, $c_{t_1,P_3,t_2,P_2} = 2$, $c_{t_1,P_3,t_2,P_4} = 6$ *(see the corresponding weighted conflict graph depicted in Figure 2.4). In this case, the "weak" inequalities (2.10) have the form:*

$$\begin{aligned} w_{t_1,t_2} &\geq 5x_{t_1,P_1} + 5x_{t_2,P_2} - 5, \\ w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 3x_{t_2,P_4} - 3, \\ w_{t_1,t_2} &\geq 2x_{t_1,P_3} + 2x_{t_2,P_2} - 2, \\ w_{t_1,t_2} &\geq 6x_{t_1,P_3} + 6x_{t_2,P_4} - 6. \end{aligned}$$

We have

$$Q_{t_1,t_2} = \text{conv}\{(1,0,1,0,5), (1,0,0,1,3), (0,1,1,0,2), (0,1,0,1,6)\}$$

and the "strong" non-dominated inequalities (2.12), found by enumerating the facets of Q_{t_1,t_2}*, read:*

$$\begin{aligned} w_{t_1,t_2} &\geq 5x_{t_1,P_1} + 2x_{t_1,P_3} + 5x_{t_2,P_2} + 3x_{t_2,P_4} - 5, \\ w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 3x_{t_1,P_3} + 2x_{t_2,P_2} + 3x_{t_2,P_4} - 3, \\ w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 2x_{t_1,P_3} + 3x_{t_2,P_2} + 3x_{t_2,P_4} - 3, \\ w_{t_1,t_2} &\geq 2x_{t_1,P_1} + 2x_{t_1,P_3} + 2x_{t_2,P_2} + 2x_{t_2,P_4} - 2, \\ w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 6x_{t_1,P_3} + 2x_{t_2,P_2} + 6x_{t_2,P_4} - 6, \end{aligned}$$

meaning $\mathcal{F}_{t_1,t_2} = \{(5,2,5,3,5), (3,3,2,3,3), (3,2,3,3,3), (2,2,2,2,2), (3,6,2,6,6)\}$.

2.2.3 The overall ILP model

To summarize, the ILP formulation that we use has objective function (2.9) and constraints (2.2), (2.5), (2.6), (2.7), and:

$$w_{t_1,t_2} \geq \sum_{P_1 \in \mathcal{P}_{t_1}} \alpha_{P_1}\, x_{t_1,P_1} + \sum_{P_2 \in \mathcal{P}_{t_2}} \beta_{P_2}\, x_{t_2,P_2} - \gamma, \qquad (t_1,t_2) \in T^2,\ (\alpha,\beta,\gamma) \in \mathcal{F}_{t_1,t_2}. \quad (2.13)$$

2.3 Solution of the LP Relaxation

As is often the case for the ILP formulations whose LP relaxations yield strong bounds on the optimal integer value, the ILP formulation of the previous section has a large number of variables and constraints. We adopt a canonical approach in which we work with a reduced current LP with all the y and w variables and a subset of the x variables, and all constraints (2.2) and (2.6) and only a subset of constraints (2.7) and (2.13). Variables and constraints are added dynamically as follows, taking into account the fact that in our case study (as well as in the other TPP case studies we are aware of) all patterns can be listed explicitly.

2.3.1 Variable pricing

We check if there are negative-reduced-cost x variables to be added to the current LP by explicitly computing all the reduced costs. This is conceptually easy but not entirely trivial since the constraints that are present in the current LP are defined only with respect to the x variables that are present. Consequently, computation of the reduced cost of a variable $x_{t,P}$ requires determining the coefficients of this variable for the constraints in the current LP. This is immediate for constraints (2.2), the coefficient being 1 for the constraint associated with train t, and (2.6), the coefficient being 1 for all constraints associated with the platform b at which pattern P stops and with instants $j \in J_b \cap [l, k]$, where $[l, k]$ is the platform occupation interval of pattern P.

As to constraints (2.7) and (2.13), there are several (in general, exponentially many) ways to extend them to include also the x variables that are not in the current LP. For the purpose of pricing, we can consider, for each variable $x_{t,P}$ and for each of these constraints, the maximum possible coefficient for the variable in an extension of the constraint.

Specifically, for each constraint (2.7), the maximum possible coefficient of variable $x_{t,P}$ is 1 if and only if either $t_1 = t$ and (t, P) is incompatible with all $(t_2, P_2) \in K$, or $t_2 = t$ and (t, P) is incompatible with all $(t_1, P_1) \in K$. Otherwise, the coefficient is necessarily 0.

Moreover, for each constraint (2.13), the coefficient of variable $x_{t,P}$ can clearly be positive only if $t_1 = t$ or $t_2 = t$. Assuming $t = t_1$, and letting $\mathcal{P}'_{t_2}$ be the set of patterns associated with variables x_{t_2,P_2} in the current LP, the maximum possible coefficient for $x_{t,P}$ in the constraint is given by

$$\min_{P_2 \in \mathcal{P}'_{t_2}} c_{t,P,t_2,P_2} + \gamma - \beta_{P_2}.$$

After the addition of the new variables, we extend the constraints (2.7) and (2.13) by computing the coefficients for these variables as follows. We consider these variables in arbitrary order and, for each of them, we set its coefficient to the maximum possible value, taking into account also the coefficients of the new variables already considered.

2.3.2 Separation of constraints (2.7)

In the sequel, let y^*, x^*, w^* be the current LP solution.

Given that all patterns associated with the same train are pairwise incompatible due to constraints (2.2), the pattern-incompatibility graph restricted to the nodes corresponding to the patterns in $\mathcal{P}_{t_1} \cup \mathcal{P}_{t_2}$ turns out to be the complement of a *bipartite* graph, with the two sides of the bipartition corresponding to the patterns in $\mathcal{P}_{t_1}$ and those in $\mathcal{P}_{t_2}$, respectively.

Therefore, separation of constraints (2.7) calls for the separation of clique inequalities in the complement of a bipartite graph, or, equivalently, to the separation of stable set

inequalities in a bipartite graph. This in turn corresponds to the determination of a maximum-weight stable set in a bipartite graph (with weight $x^*_{t_i,P}$ for each node (t_i, P), $i = 1, 2$), which is well-known to be a minimum s,t-cut problem on a directed network with source s, terminal t, and the other nodes corresponding to the nodes in the bipartite graph, which can be solved efficiently by any maximum-flow code.

Proposition 3. *Constraints (2.7) can be separated by computing $|T^2| = O(|T|^2)$ maximum flows in a network with $O(p_{\max})$ nodes, where $p_{\max} := \max_{t\in T} |\mathcal{P}_t|$.*

2.3.3 Separation of constraints (2.13)

The separation of constraints (2.13) is done by a sort of "polyhedral brute force", given that, for each pair of trains t_1, t_2, the number of vertices in Q_{t_1,t_2} is "small". Specifically, since Q_{t_1,t_2} has $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ vertices and lies in $\mathbb{R}^{|\mathcal{P}_{t_1}|+|\mathcal{P}_{t_2}|+1}$, we can separate over it by solving the following LP with $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ variables and $|\mathcal{P}_{t_1}| + |\mathcal{P}_{t_2}| + 1$ constraints.

Recall the form of the vertices of Q_{t_1,t_2} given in its definition (2.11). We have that the vector $((x^*_{t_1,P_1})_{P_1\in\mathcal{P}_{t_1}}, (x^*_{t_2,P_2})_{P_2\in\mathcal{P}_{t_2}}, w^*_{t_1,t_2})$ belongs to Q_{t_1,t_2} if and only if it can be expressed as a convex combination of its vertices, i.e., letting λ_{P_1,P_2} be the multiplier associated with vertex $(e_{P_1}, e_{P_2}, c_{t_1,P_1,t_2,P_2})$, if and only if there exists a solution to the linear system:

$$x^*_{t_1,P_1} = \sum_{P_2\in\mathcal{P}_{t_2}} \lambda_{P_1,P_2}, \qquad P_1 \in \mathcal{P}_{t_1}, \tag{2.14}$$

$$x^*_{t_2,P_2} = \sum_{P_1\in\mathcal{P}_{t_1}} \lambda_{P_1,P_2}, \qquad P_2 \in \mathcal{P}_{t_2}, \tag{2.15}$$

$$1 = \sum_{P_1\in\mathcal{P}_{t_1}} \sum_{P_2\in\mathcal{P}_{t_2}} \lambda_{P_1,P_2}, \tag{2.16}$$

$$w^*_{t_1,t_2} = \sum_{P_1\in\mathcal{P}_{t_1}} \sum_{P_2\in\mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ \lambda_{P_1,P_2}, \tag{2.17}$$

$$\lambda_{P_1,P_2} \geq 0, \qquad P_1 \in \mathcal{P}_{t_1},\ P_2 \in \mathcal{P}_{t_2}. \tag{2.18}$$

Applying Farkas' Lemma, and letting α_{P_1}, β_{P_2}, γ' and ε be the dual variables associated with constraints (2.14), (2.15), (2.16) and (2.17), respectively, we have that the linear system (2.14)–(2.18) has a solution if and only if the optimal value of the following LP is zero:

$$\max \sum_{P_1\in\mathcal{P}_{t_1}} \alpha_{P_1}\ x^*_{t_1,P_1} + \sum_{P_2\in\mathcal{P}_{t_2}} \beta_{P_2}\ x^*_{t_2,P} + \gamma' + \varepsilon\ w^*_{t_1,t_2} \tag{2.19}$$

subject to

$$\alpha_{P_1} + \beta_{P_2} + \gamma' + \varepsilon\ c_{t_1,P_1,t_2,P_2} \leq 0, \qquad P_1 \in \mathcal{P}_{t_1},\ P_2 \in \mathcal{P}_{t_2}. \tag{2.20}$$

In other words, the vector does not belong to Q_{t_1,t_2} if and only if the optimal value of LP (2.19)–(2.20) is positive (in fact, infinity). Given that we are interested in separating constraints of the form (2.13), it is easy to check that we can replace "=" by "≥" in constraints (2.16) and (2.17), leading to $\gamma', \varepsilon \leq 0$, and then add the normalization condition $\varepsilon = -1$ and replace γ' by $\gamma := -\gamma'$; in this way the objective function (2.19) calls exactly for the determination of the constraint (2.13) that is violated by the largest amount. Then, for each $(t_1, t_2) \in T^2$, we separate constraints (2.13) by solving LP (2.19)–(2.20) after the small changes above.

Proposition 4. *Constraints (2.13) can be separated by solving* $|T^2| = O(|T|^2)$ *LPs with* $O(p_{\max})$ *variables and* $O(p_{\max}^2)$ *constraints, where* $p_{\max} := \max_{t \in T} |\mathcal{P}_t|$.

2.4 The Overall Method

In this section we describe our solution approach to TPP, whose main component is the solution of the LP relaxation of the ILP model of Section 2.2 by the method described in Section 2.3.

2.4.1 A branch-and-bound method

Our overall method is a branch-and-bound method in which branching is aimed at quickly finding a "good" feasible solution. This makes it essentially a canonical *diving* heuristic algorithm that, rather than terminating at the end of the "dive", continues as a regular branch-and-bound method until optimality is proved (or the time limit is reached).

Specifically, given the optimal LP solution y^*, x^*, w^*, if x^* is integer this is also the optimal ILP solution of the current branch-and-bound problem (defined as the original ILP with the addition of the branching constraints, see below). Otherwise, we select the variable $x_{t,P}$ which is not fixed by branching constraints and whose value $x^*_{t,P}$ is closest to 1 (*possibly* it is 1). We generate two problems by imposing, respectively, the *branching constraints* $x_{t,P} = 1$ and $x_{t,P} = 0$, and explore the first problem generated before the second, in a *depth-first* fashion. (Note that, if $x^*_{t,P} = 1$, there is no need to solve again the LP relaxation of the first problem.) The first backtracking occurs when we have an integer solution for a problem for which the branching constraints have fixed $x_{t,P} = 1$ for the x components with largest LP value encountered. Until this backtracking, the method is a basic textbook diving heuristic algorithm.

The solution of the LP relaxation in the problems after the original *root* one is still carried out by pricing and separation, which makes the method a branch-and-cut-and-price one.

2.4.2 Heuristic methods

In order to try to obtain good feasible solutions within computing times shorter than the ones of the diving heuristic algorithm of the previous section, we also implemented a wide number of relatively fast heuristic methods based on the solution of the LP relaxation. Despite the wide number of attempts, based on criteria that turned out to be successful for a few analogous problems, we did not really succeed in improving the results of the diving heuristic algorithm of the previous section. Namely, the results we obtained were generally of much worse quality, and still required the solution of the LP relaxation, so also the running times were not much faster. For this reason, we do not illustrate these heuristic methods here in detail nor we report the associated computational results.

Another possible way to obtain solutions within shorter computing times is to stick to the diving heuristic algorithm, but to use a much simpler objective function in which we only minimise the number of dummy platforms used. In this way the quadratic term in the objective function is not present, i.e., the w variables along with constraints (2.13) for the linearisation are not needed and the model turns to be much smaller. The solutions obtained, evaluated with the original objective function, turn out to be of reasonable quality for the instances of our case study.

2.4.3 Implementation details

For the root problem, we initialize the current LP with all the y variables, no w variable (see below), and the x variables corresponding to the $|T|$ patterns selected by an elementary greedy heuristic method, which considers the trains by decreasing values of the train priority (defined for our case study, see Section 2.1.3) and, for each train, chooses the pattern that is compatible with the patterns already chosen and leads to the smallest increase in the objective function. Moreover, our initial LP has all constraints (2.2) and (2.6) and no constraints (2.7) and (2.13). Each variable w_{t_1,t_2} is introduced in the LP as soon as one constraint (2.13) associated with it is added.

Given the solution of the current LP, we first perform pricing by finding, for each train, the pattern with the most negative reduced cost. If any patterns are found, we add them to the current LP and solve it by primal simplex. Otherwise, i.e., if there is no pattern with negative reduced cost, we separate constraints (2.7) by solving, for each pair $(t_1, t_2) \in T^2$, the minimum s,t-cut problem mentioned in Section 2.3.2 as an LP, given that these cut problems turn out to be computationally very easy and their overall solution time is negligible compared to the LP solution time. If any violated constraints (2.7) are found, we add them to the current LP and solve it by dual simplex. Otherwise, we separate constraints (2.13) by solving, for each pair $(t_1, t_2) \in T^2$, the LP defined in Section 2.3.3. If any violated constraints (2.13) are found, we add them to the current LP and solve it by dual simplex. Otherwise, the LP for the current branch-and-bound problem is solved.

2.5 Experimental Results

In this section, we illustrate the results obtained for the instances of our case study. Our method was implemented in ANSI C and tested on a PC Pentium 4, 3.2 GHz, with a 2 GB RAM, using ILOG CPLEX 9.0 as LP solver. All the computing times are expressed in seconds.

2.5.1 The instances

instance	station name	$\|T\|$	$\|B\|$	$\|D\|$	$\|\mathcal{R}\|$	# inc.	$g_d^{\max}$
PA C.LE.	Palermo Centrale	204	11	4	64	1182	3
GE P.PR.	Genova Piazza Principe	127	10	4	174	7154	4
BA C.LE.	Bari Centrale	237	14	5	89	1996	4
MI C.LE.	Milano Centrale	215	24	7	312	29294	5

Table 2.1: Instances in our case study.

Table 2.1 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 ($|\mathcal{R}|$), the number of pairs of incompatible paths (# inc.), and the maximum travel time ($g_d^{\max} := \max_{d \in D} g_d$).

As the results in the following show, the instance MI C.LE turns out to be much harder to solve than the other three instances. This is apparently in contrast with the fact that the

number of trains in the instance is approximately the same as for the instances PA C.LE and BA C.LE, and that the number of platforms is larger, which would suggest that assigning these trains to platforms is easier. In fact, the difficulty of MI C.LE is due to the much larger number of paths and incompatible path pairs. Due to this difficulty, we will present separately the results for the first three instances and for MI C.LE.

2.5.2 Results for PA C.LE, GE P.PR, BA C.LE

instance	π	curr	LP		first		best	
			value	time	value	time	value	time
PA C.LE.	0	749012	334038	27	349037	48	339039	56
PA C.LE.	1	410139	10159	56	120155	96	120155	96
PA C.LE.	2	380182	10159	53	10176	86	10176	86
PA C.LE.	3	11431	10159	54	10172	106	10172	106
GE P.PR.	0	745000	306020	67	306020*	115	306020*	115
GE P.PR.	1	705005	147069	192	147087	490	147080	825
GE P.PR.	2	458065	8116	274	8121	499	8116*	4617
GE P.PR.	3	336340	8116	572	8121	1057	8116*	13647
GE P.PR.	4	8692	8116	434	8126	866	8116*	1040
BA C.LE.	0	1576300	653264	122	808255	350	808255	350
BA C.LE.	1	1398330	373486	123	438685	642	438685	642
BA C.LE.	2	1197485	128896	199	148867	542	148867	542
BA C.LE.	3	838235	8885	216	8924	656	8924	656
BA C.LE.	4	11290	8877	198	8912	880	8911	52943

Table 2.2: Results of the branch-and-bound method for the instances PA C.LE, GE P.PR, BA C.LE.

In Table 2.2 we compare the solution obtained by a (computationally very fast) heuristic method currently used by Rete Ferroviaria Italiana with the best integer solution produced by our branch-and-bound approach with a time limit of 1 day (24 hours), recalling that TTP is a planning problem to be solved each time a new timetable is released.

We tested all the possible integer values of the dynamic threshold π defined in Section 2.1.3. These values range from 0, in which case the occupation of incompatible paths at the same time is forbidden and the quadratic part in the objective function vanishes, to $g_d^{\max}$, in which case two patterns are incompatible if and only if they occupy the same platform at the same time, whereas simultaneous occupation of incompatible paths does not affect the problem feasibility but only the quadratic part in the objective function.

In Table 2.2, we report the value of π, the solution value found by the heuristic method currently used by Rete Ferroviaria Italiana (*curr*), and the (rounded-up) value and the associated computing time for the LP relaxation at the root problem (*LP*), for the first feasible solution found by our "diving" branch-and-bound method (*first*), and for the best feasible solution found by branch-and-bound within the time limit (*best*); a "*" means that the solution found is optimal.

The table shows that in all cases our approach was able to improve significantly over

the current heuristic solution, in most cases finding the best solution, or a solution of value very close to the best one, after a fairly small running time (some minutes). In 4 out of 14 cases the best solution found is provably optimal (and the first solution found is very close to the optimal one), in other 5 cases the relative gap between the first solution value and the LP lower bound is less than 1%, whereas in the remaining 5 cases the gap is not negligible, ranging from about 1.5% to the huge gap for PA C.LE. with $\pi = 1$, for which we do not know if the dummy platform that is used by the best solution found is really necessary (see also below).

instance	π	curr	LP		first		best		
			value	time	value	time	value	time	original
PA C.LE.	0	3	2	26	2*	38	2*	38	541101
PA C.LE.	1	2	0	28	1	45	1	45	231242
PA C.LE.	2	2	0	10	0*	10	0*	10	11422
PA C.LE.	3	0	0	7	0*	7	0*	7	11431
GE P.PR.	0	3	2	39	2*	39	2*	39	577037
GE P.PR.	1	3	1	97	1*	97	1*	97	369139
GE P.PR.	2	2	0	70	0*	91	0*	91	9557
GE P.PR.	3	2	0	22	0*	22	0*	22	8716
GE P.PR.	4	0	0	8	0*	8	0*	8	8692
BA C.LE.	0	5	2	214	3	354	3	354	1107800
BA C.LE.	1	4	2	48	2*	87	2*	87	743701
BA C.LE.	2	4	1	39	2	55	1*	62	396841
BA C.LE.	3	3	0	60	0*	60	0*	60	15053
BA C.LE.	4	0	0	9	0*	9	0*	9	11290

Table 2.3: Results of the branch-and-bound method for the instances PA C.LE, GE P.PR, BA C.LE if the objective is the minimisation of the number of dummy platforms.

Table 2.3 shows the same comparison as Table 2.2, but on the simpler objective function where we only minimise the number of dummy platforms used, as discussed above. Note that for GE. P.PR. we find a provably optimal solution in all cases, and for both PA C.LE and BA C.LE in all cases except one (with a gap of one unit). In particular, as anticipated above, we do not know if one dummy platform is necessary for PA C.LE. with $\pi = 1$. All these solutions are found within minutes and are often significantly better than the solutions found by the heuristic method currently in use. In column *original* we report, for each best solution found, the corresponding value of the original objective function. This shows that the latter is notably better than the current heuristic one but also notably worse than the best value reported in Table 2.2 (although with smaller computing times).

2.5.3 Results for MI C.LE

For the instance MI C.LE, even only solving the LP relaxation turned out to be computationally challenging. For this reason, in order to speed up the LP solution, we determined some early termination criteria. Specifically, it is well known that, in case there is no pattern with negative reduced cost, we are at the end of a pricing phase and the current LP value is

a lower bound on the solution value of the complete LP. First of all, in case this lower bound did not change (increase, due to the constraints added) for p consecutive pricing phases (with $p = 3$ in our implementation), we terminate the LP solution. Moreover, we terminate the LP solution also in case the current LP value did not change (increase) after having added violated constraints (2.7) or (2.13), noting that in this case this value is not necessarily a valid lower bound for the complete LP since there may be patterns with negative reduced cost. In any case, these early termination criteria are not applied in case the current LP solution is integer and not feasible, to avoid branching starting from an integer infeasible solution.

instance	π	curr	LP		first		best	
			value	time	value	time	value	time
MI C.LE.	0	2739337	1390514	1022	2094441	8935	2094441	8935
MI C.LE.	1	2553299	1324391	2495	1924914	21990	1924814	22016
MI C.LE.	2	2490357	1153469	2658	1826676	36216	1826581	37764
MI C.LE.	3	2300913	143091	30316	1039227	32291	1029329	32323
MI C.LE.	4	1824743	20432	36713	-	-	-	-
MI C.LE.	5	455504	20424	25265	22339	420406	22339	420406

Table 2.4: Results of branch-and-bound for the instance MI C.LE.

Table 2.4 reports the results obtained by the branch-and-bound algorithm for the various values of π, after having introduced the early LP termination criteria mentioned above. For $\pi = 0, 1, 2, 3$, we can find in a few hours a solution whose value is notably better than the current one, although the gap with respect to the LP lower bound is fairly large, mainly because our heuristic solutions use more dummy platforms than those corresponding to the lower bound determined by the LP relaxation (see below). On the other hand, for the cases $\pi = 4$ and $\pi = 5$ no solution was found within the 1-day time limit. By extending this time limit to 5 days, a solution was found for $\pi = 5$, which turns out to be close to the optimal one, whereas no solution was found for $\pi = 4$. A feasible solution for the latter case is found within a few hours by considering the simpler objective function, as discussed in the following.

instance	π	curr	LP		first		best		
			value	time	value	time	value	time	original
MI C.LE.	0	14	7	455	12	28280	11	29693	2293575
MI C.LE.	1	13	7	298	12	19073	10	22026	2113344
MI C.LE.	2	13	7	240	10	6356	9	6524	1825695
MI C.LE.	3	12	1	1355	7	258011	7	258011	1192697
MI C.LE	4	10	0	955	4	23834	2	24181	311578
MI C.LE	5	3	0	78	0*	78	0*	78	30619

Table 2.5: Results of the branch-and-bound method for the instance MI C.LE if the objective is the minimisation of the number of dummy platforms.

Table 2.5 is the counterpart of Table 2.3 for the instance MI C.LE, still with the early termination criteria in the LP solution. All solutions are much better than the current

heuristic ones, even if we consider their values according to the original objective function. With the exception of the case $\pi = 3$, for which the running time is very large, the solutions are found within approximately 8 hours. For the case $\pi = 4$, we find a feasible solution, for which the gap with respect to the LP lower bound is fairly large (both in the simplified and in the original objective function value) since we do not know if two dummy platforms are really necessary.

2.6 Conclusions

The main practical impact of our approach, if applied in place of the simple heuristic method currently in use, is to extend the current "capacity" of the stations considered, using a smaller number of platforms for the current trains and then allowing new trains to stop at the station (if the capacity along the lines associated with the directions allows this).

The relatively wide gaps encountered in some cases, between the best heuristic value that we can find and the LP lower bound, should motivate further research on the problem. Moreover, it would be interesting to have (much) faster heuristic methods producing solutions of comparable quality in order to be able to apply them if a disrupted schedule has to be adjusted within relatively short time, especially for cumbersome instances such as MI C.LE.

2.7 Appendix 1

For sake of completeness we present a first, naive model that we used. This model could not solve the complete instances of RFI, even if it managed to find optimal or nearly optimal solutions for some smaller cases. Still, its significance was completely overshadowed by the model presented in the previous sections. Notice that the formulation is based on slightly different requirements, as we will show in the sequel.

2.7.1 A naive ILP Formulation

Recall the ILP formulation defined by objective function (2.9) and constraints (2.2), (2.5) (2.6), (2.7), and (2.13). Roughly, the model contains two types of constraints. Those that account for feasibility:

- assignment constraints (2.2)
- incompatibility constraints on the platforms (2.6)
- incompatibility constraints on the paths (2.7)

and those that link the patterns with the variables needed to define the objective function:

- linking between $x_{t,P}$ and y_b (2.6)
- linking between $x_{t,P}$ and w_{t_1,t_2}, to achieve linearisation (2.13)

Notice that the purpose of constraints (2.6) is twofold: they forbid conflicts over the platforms and link the variables involved ($x_{t,P}$ and y_b) at the same time. The same rationale applies to the naive model, but with a less concise structure. We still use binary variables $x_{t,P}$, y_b together with the assignment constraints (2.2) to give each train a pattern. An additional requirement, proposed by RFI at the very beginning, was to add a term in the objective function to penalise the maximum number of platforms occupied simultaneously (*bottle neck* constraint). Similarly, each minute of conflict over any path had to be penalised. For this reason two different types of binary variables were added: $\alpha_{b,j}$, to map platform occupation ($b \in B$, set of platforms in the station) for every instant j, and $z_{t_1,t_2,j}$ to express a path conflict at instant j between the pair of trains t_1 and t_2. So the main feature of the naive model is *time dependency*, since we introduced two types of auxiliary variables for every instant (minute) of the optimisation time horizon.

Here is the complete model:

$$\min \sum_{b\in B} c_b y_b + \sum_{t\in T} \sum_{P\in\mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{(t_1,t_2)\in T^2} c_{t_1,t_2} \sum_{j\in J} z_{t_1,t_2,j} + \beta \tag{2.21}$$

subject to

$$\sum_{P\in\mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{2.22}$$

$$\alpha_{b,j} - \sum_{t\in T} \sum_{P\in\mathcal{P}_t} a_{b,j}^{t,P} x_{t,P} \geq 0, \qquad b \in B, j \in J, \tag{2.23}$$

$$y_b - \alpha_{b,j} \geq 0, \qquad b \in B, j \in J, \tag{2.24}$$

$$\sum_{j \in F_h} z_{t_1,t_2,j} \leq \pi, \qquad F_h \in F_\pi, (t_1,t_2) \in T^2, \tag{2.25}$$

$$z_{t_1,t_2,j} \geq \sum_{(t_1,P_1) \in K} x_{t_1,P_1} + \sum_{(t_2,P_2) \in K} x_{t_2,P_2} - 1, \qquad (t_1,t_2) \in T^2, j \in J, K \in \mathcal{K}_{t_1,t_2,j}, \tag{2.26}$$

$$\beta - \sum_{b \in B} \alpha_{b,j} \geq 0, \qquad j \in J, \tag{2.27}$$

$$y_b, x_{t,P}, z_{t_1,t_2,j}, \alpha_{b,j} \in \{0,1\}, \qquad b \in B,\ t \in T,\ P \in \mathcal{P}_t, (t_1,t_2) \in T^2, j \in J. \tag{2.28}$$

Auxiliary variables ($\alpha_{b,j}$ and $z_{t_1,t_2,j}$) are used to express both feasibility of the pattern selection over the station resources (platforms and paths) and the objective function terms. Thus, two types of constraints are needed for each of them:

- linking with the pattern variables (2.23), (2.26)
- feasibility for a specific resource, either a platform or a path (2.24), (2.25)

Time is discretised in minutes as usual and J is the set of minutes in a day. Notice that in this formulation we do not have clique constraints that forbid the choice of subsets (cliques) of pairwise incompatible patterns (2.7). In fact feasibility on the platforms is attained indirectly by avoiding overlapping instant by instant (2.24). Binary matrix $a^{t,P}_{b,j}$ tells if pattern P of train t occupies platform b at instant j. Whereas, we achieve feasibility on the paths by observing that time windows with more than π consecutive instants of conflict on the paths must be forbidden. The possibility of having path conflicts under the threshold π is expressed by constraints (2.25). In fact, F_π is a set of time windows of length $\pi + 1$ that forbid $\pi + 1$ consecutive instants of conflicts on a path. Constraints (2.25) are expressed as sliding time windows starting at instant h and stretching on the subsequent $\pi+1$ instants for every pair of trains. The linking between $x_{t,P}$ and $z_{t_1,t_2,j}$ variables is obtained strengthening the classical linearisation technique (2.8) using cliques. In other words, in a similar fashion to the pattern incompatibility graph described in Section 2.2, we construct a *time-pattern* conflict graph $G(t_1,t_2,j)$ (see Figure 2.5), whose vertices are patterns of trains t_1 and t_2, connected by an edge if at instant j, the two patterns cause a conflict. Hence, in order to bound $z_{t_1,t_2,j}$ from below, we look for cliques of conflicting patterns on this graph (see Figure 2.6 for an example of clique of conflicting patterns). The idea boils down to a clique strengthening method of the basic linearisation constraints (2.8). So, for every pair of trains and for every instant, the set $\mathcal{K}_{t_1,t_2,j}$ contains all maximal cliques on the conflict graph $G_{t_1,t_2,j}$. Finally *bottle neck* constraints on the maximum number of platforms occupied simultaneously are added (2.27) in order to keep the train station area as free as possible. As said, this constraint is not present in the new formulation, since the minimisation of the number of platforms used already hedges against the overall platform occupation.

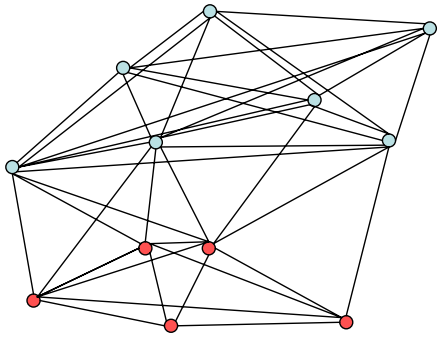

Figure 2.5: Conflict graph $G(t_1, t_2, j)$.

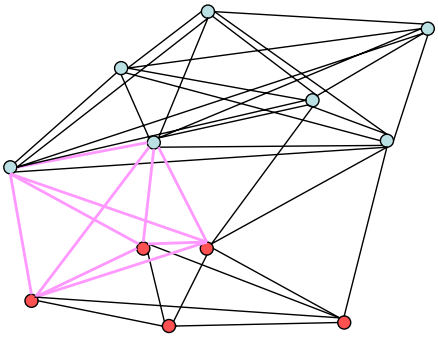

Figure 2.6: Clique of conflicting patterns on $G(t_1, t_2, j)$.

2.7.2 Solution process

The number of patterns in a medium sized instance is pretty large, hence the solution of the LP required a column generation approach. The pricing problem for $x_{t,P}$ variables is performed by enumerating all feasible patterns for every train and computing the corresponding reduced cost. The $z_{t_1,t_2,j}$ and $\alpha_{b,j}$ variables are also handled dynamically. Platform variables y_b are inserted in the model at the beginning. Similarly, separation is necessary in order to handle constraints (2.25) and (2.26). Constraints (2.25) are separated via enumeration, by considering all time windows for every pair of trains. Whereas, the separation problem for constraints (2.26) corresponds to a maximum weight clique problem (MWCP) on the corresponding conflict graph. In order to compute maximum weight cliques, we used a two phase approach: at first, a fast heuristic procedure for the MWCP was run, if it was able to find a violated constraint this was added to the model, if not, the exact separation routine had to be used. Notice that the exact procedure was still rather fast since the complementary of the given conflict graph is bipartite, hence the task can be performed in polynomial time solving a max-flow min-cut problem as described in 2.3.2.

A said, the main feature of this model is *time dependency*, in fact both constraints on path and platform conflicts are expressed minute by minute, and this is also the main drawback. The LP solution process was rather slow and, once inserted in a branch-and-bound framework,

it was not able to tackle complete real-world instances.

2.7.3 Experimental Results

The naive model could only solve some small instances extracted from the original ones. Palermo C.Le. could be solved up to 100 out of 204 trains simultaneously, Bari C.Le could cope only with 25 trains. In the following tables we report for each instance the number of trains, the value of the threshold, the LP solution value, the best ILP solution found within a time limit of two hours and the corresponding time (if the time limit was not reached).

instance	trains	π	curr	LP	best ILP	time
PA C.LE.	10	0	5050	4044	4044	5
PA C.LE.	10	1	5056	4044	4044	5
PA C.LE.	10	2	5056	4044	4044	5
PA C.LE.	25	0	146070	126072	126072	36
PA C.LE	25	1	136070	116587	117086	190
PA C.LE	25	2	127076	7104	7105	25
PA C.LE	50	0	438115	248865	360112	-
PA C.LE	50	1	269149	117162	120165	-
PA C.LE	50	2	269172	10173	10185	-
PA C.LE	100	0	539116	214138	280134	-
PA C.LE	100	1	319259	124824	140189	-
PA C.LE	100	2	319294	10255	10257	-

Table 2.6: Results of the branch-and-bound method on the old model for instance PA C.LE.

instance	trains	π	curr	LP	best ILP	time
BA C.LE.	10	0	123040	113542	114040	-
BA C.LE.	10	1	123040	103082	104065	-
BA C.LE.	10	2	113065	3076	3100	-
BA C.LE.	25	0	284060	145061	145061	2400
BA C.LE	25	1	147071	111104	116102	-
BA C.LE	25	2	145071	6117	6117	780

Table 2.7: Results of the branch-and-bound method on the old model for instance BA C.LE.

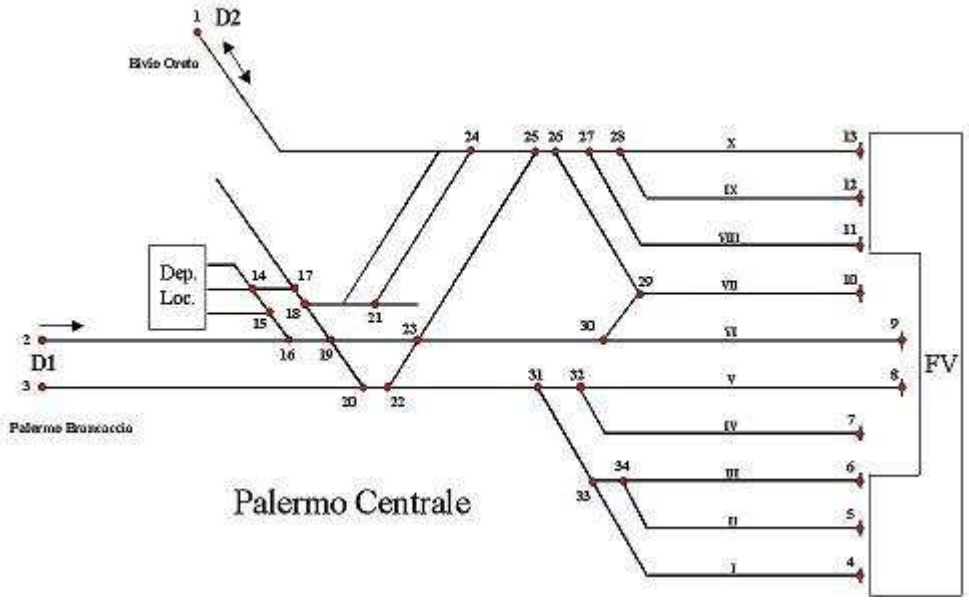

Figure 2.7: The layout of Palermo Central Station (BA C.LE).

2.8 Appendix 2

Here are a few more train station layouts for the instances considered: Palermo C.Le. and Genova P.Princ.

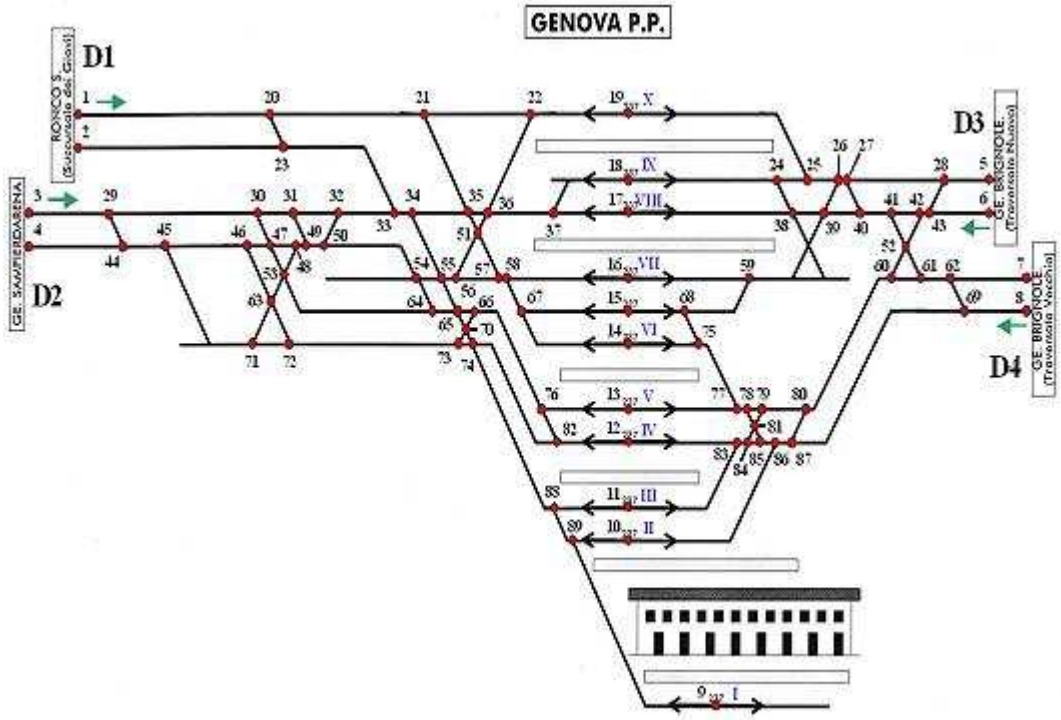

Figure 2.8: The layout of Genova Porta Principe Station (GE P.Princ.).

Chapter 3

Robust Optimisation and Recoverable Robustness

In this chapter we will first give an overview on *classical* Robust Optimisation and some related concepts, and then move on to the *new* idea of *Recoverable Robustness*.

It is well known that in practical optimisation problems, such as railway applications, information is always imperfect. In fact, data are in general subject to changes and some of them may even be unknown at the moment of planning (optimisation). On the other hand, standard classical optimisation assumes that all data are completely known in advance with infinite precision. Hence, the need to develop theoretical and algorithmic methods to construct solutions that can still be significant and useful when the original data are affected by some level of inaccuracy. The first interest and efforts towards the definition of robust optimisation models date back to 1970s, with the works by Soyster [33] lately extended by Bertsimas and Sim in 2004 [5]. Recently, in the last three years, a new concept for robust optimisation, namely Recoverable Robustness, has been developed by Liebchen et al. [28], [34]. The idea is an extension of the classical concept, devised by taking a closer look at the practical needs in many real life contexts. In fact, by incorporating the idea of *recovery*, i.e. the possibility to change and adjust the original solution according to the needs of the newly realized data, we are able to deal with real world problems in a much more flexible and effective way.

3.1 A bit of history

3.1.1 Classic Robust optimisation

The main features of Robust Optimisation are:

- Computational efficiency of many robust models, w.r.t. stochastic models.
- The possibility to rely always (not just on average) on robust solutions, within the limits of the scenarios considered. In fact, expectations are not the right objective in many applications.
- Distributions, which are usually difficult to handle, are not needed.

On the other hand, the main drawback is that classical robust optimisation is incapable of finding solutions that are robust if a certain amount of recovery is granted. In other words, solutions are over conservative, because they must be *fixed* prior the realisation of the uncertain data and must be feasible for *all* scenarios of the (possibly limited) scenario set.

The first step taken by Soyster [33] consisted in a linear optimisation model able to construct a solution that is feasible for all data belonging to a convex set. The main weakness of such a formulation is the extreme over conservativeness, in fact too much of optimality for the nominal problem is lost in order to ensure robustness.

$$\begin{aligned} & \max c'x \\ \text{s.t.} \quad & \sum_{j=1}^{n} A_j x_j \leq b \quad \forall A_j \in \mathcal{K}_j, j = 1, ..., n \\ & x \geq 0, \end{aligned}$$

where the uncertainty sets $\mathcal{K}_j$ are convex. For a system of linear constraints $Ax \leq b$, where A is unknown data in some range, the Soyster model assumes the worst value simultaneously for each entry. Indeed, the former formulation is equivalent to the following:

$$\begin{aligned} & \max c'x \\ \text{s.t.} \quad & \sum_{j=1}^{n} \bar{A}_j x_j \leq b \\ & x \geq 0, \end{aligned}$$

where $\bar{a}_{ij} = \sup_{A_j \in \mathcal{K}_j}(A_{ij})$.

To overcome this problem Ben-Tal and Nemirovski [2],[3],[4], and El-Ghaui et al. [18],[19] proposed less conservative models by considering ellipsoidal uncertainties that require to solve conic quadratic programs. Clearly, the drawback of these approaches is the necessity to deal with nonlinear, although convex, models that are particularly demanding computationally.

The brilliant breakthrough by Bertsimas and Sim [5] was to devise a new approach that retains the advantages of a linear framework, but also offers control on the degree of over conservativeness for every constraint. In fact, their model guarantees that a solution is feasible with respect to constraint i if less then Γ_i uncertain coefficients change at the same time. Moreover, the robust counterparts always remain linear problems, hence they can be easily generalized to discrete optimisation problems. More precisely, they consider the ith constraint of the problem: $a_i x \leq b_i$. Let J_i be the set of coefficients a_{ij}, $j \in J_i$ that are subject to parameter uncertainty, i.e., $\tilde{a}_{ij}$, $j \in J_i$ take values according to a symmetric distribution with mean equal to the nominal value a_{ij} in the interval $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$. For every i, they introduce a parameter Γ_i, not necessarily integer, that takes values in the interval $[0, |J_i|]$. The role of parameter Γ_i is to adjust the robustness of the proposed method against the level of conservatism of the solution. As we pointed out, it is very unlikely that all of the $a_{ij}, j \in J_i$ will change simultaneously. Thus, the goal of the method is to protect against all cases where up to $\lfloor \Gamma_i \rfloor$ of these coefficients are allowed to change, and one coefficient a_{it} changes by $(\Gamma_i - \lfloor \Gamma_i \rfloor)\hat{a}_{it}$. In other words, only a subset of the coefficients will change and adversely affect the solution. The approach they developed has the property that if aberrations are maintained within these limits, the solution will be feasible *deterministically*.

They also proved that even if more than $\lfloor \Gamma_i \rfloor$ change, the solution will still be feasible *with very high probability.*

Their original formulation is the following:

$$\max c'x$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij}x_j + \max_{\{S_i \cup \{t_i\} | S_i \subseteq J_i, |S_i| = \lfloor \Gamma_i \rfloor, t_i \in J_i \setminus S_i\}} \left\{ \sum_{j \in S_i} \hat{a}_{ij} y_j + (\Gamma_i - \lfloor \Gamma_i \rfloor) \hat{a}_{it_i} y_t \right\} \leq b_i \quad \forall i,$$

$$-y_j \leq x_j \leq y_j \quad \forall j,$$

$$l \leq x \leq u,$$

$$y \geq 0.$$

If Γ_i is chosen as an integer, the ith constraint is protected by $\beta_i(x, \Gamma_i) = \max_{\{S_i | S_i \subseteq J_i, |S_i| = \Gamma_i\}} \left\{ \sum_{j \in S_i} \hat{a}_{ij} |x_j| \right\}$. Note that when $\Gamma_i = 0$, $\beta_i(x, \Gamma_i) = 0$, the constraints are equivalent to those of the nominal problem. Likewise, if $\Gamma_i = |J_i|$, we are back to Soyster's method. Therefore, by varying $\Gamma_i \in [0, |J_i|]$, we have the flexibility of adjusting the robustness of the method against the level of conservatism of the solution. The authors then prove the equivalence of the afore mentioned model with a linear program. Hence, their method manages to maintain linearity for the robust counterpart of the nominal model.

So, for a given system of linear constraints $Ax \leq b$, where A and b are unknown data in some range, the Soyster model assumes the worst value simultaneously for each entry, while the Bertsimas and Sim approach manages an *horizontal integration* of the aberrations in the input data. In fact, they consider one single constraint $a_i^T x \leq b_i$ at a time and bound the uncertainty on the input data. The model stays tractable and increases substantially the quality of the solutions, making them less conservative, though robust.

Hence, classical robust models argue row-wise or horizontal. Unfortunately, this can be rather awkward in many railway application problems, which are often modeled as networks. In a network, horizontal integration has no effect. In fact in a network, inequalities like $a_i^T x \leq b_i$ represent an arc, thus they only have a single random entry each. For this reason bounding joint aberrations per row has no effect, since there is exactly one uncertain parameter in every row. Such problems require a *vertical integration*: bounding the uncertain parameters in *all* rows by a limited joint aberration is a key to less conservative solutions. Unfortunately classical robustness cannot manage this, because it relies on the fact the every constraint in every scenario must be fulfilled.

Another pivotal feature of Robust Optimisation (also called *strict* robustness) is that once computed, the solution is *never changed*, not even when the realised scenario becomes known. Still, small infeasibilities may be corrected and the solution adjusted according to the realised scenario. If such corrections are ignored the solutions we obtain may get unacceptably expensive or over conservative.

Recoverable Robustness moves forward in both directions: it allows for constraints to be violated before the *recovery* takes place and imposes a bound on the recovery of all constraints in each scenario (vertical integration). *Light Robustness*, as we will see, is a heuristic approach to this idea.

3.1.2 Stochastic Programming

Stochastic programming, for a complete treatment see [7], splits the set of constraints of the model into two: the deterministic part and the stochastic part, whose coefficients have a stochastic nature since they depend on the scenario according to a given distribution. The decisions taken in the first stage, ignoring the stochastic part of the model, will reflect some costly recourse actions in case indeterminacy occurs. A typical optimisation objective for this two-stage strategy is to minimise the total expected cost:

$$\min_{x \in X} c'x + E[Q(x, \xi(\omega))]$$

where x denotes the first-stage decision whose feasibility set is X, $\omega \in \Omega$ denotes a scenario that is unknown when the first-stage decision x has to be made, and $Q(x, \xi(\omega))$ represents the optimal value of the second-stage recourse problem corresponding to first-stage decision x and parameters $\xi(\omega)$. Hence, the idea of Recoverable Robustness to weight recovery costs against planning costs is somehow reminiscent of two-stage stochastic programming. In the stochastic setting first stage decisions are taken without full information, whereas in the second stage we achieve feasibility by second stage decisions that come at a higher cost. This is rather similar to the Recoverable Robust setting, except that the stochastic model of imperfect information contains a probability distribution for the scenario set. Such an approach requires the availability of distributions and especially it requires to handle them. Moreover, the expected cost may not be the right measure for the quality of the solution. On the other hand, strict robustness would calculate first stage cost only and require the first stage decision to be feasible for every scenario. Drawbacks on both sides are overcome by Recoverable Robustness, since it allows recovery and at the same time gives a guaranteed (upper bound) for its costs, without relying on distributions.

3.2 Light Robustness

The idea of light robustness rests on the observation that, in a sense, robustness is about allowing enough slack on the constraints of the problem: for a complete treatment refer to Fischetti et al. [21]. Thus, given

$$\max \left\{ c^T x : Ax \leq b, x \geq 0 \right\}$$

the Light Robust counterpart is the following:

$$\min f(\gamma)$$

$$Ax + \beta - \gamma \leq b$$

$$c^T x \geq (1 + \alpha) z^*$$

$$x \geq 0$$

$$0 \leq \gamma \leq \beta$$

where β_i is a parameter giving the desired protection level (or slack) on constraint i, and γ_i is a decision variable giving the corresponding unsatisfied slack. The objective function is to minimise a given function f of the variables γ. Moreover there is a bound (controlled

by parameter α) on the efficiency loss due to the increased robustness of the solution. In other words, we can interpret γ variables as a way to allow some recovery actions when the given solution does not meet the required level of protection on a constraint. Hence, Light Robustness is a heuristic method to achieve recoverable robust solutions via vertical integration of data abberations, in fact it seeks to minimise a function of all local violations of constraints, which is a way to allow (at some cost) a local recovery of the constraint. Still, the application may not allow the local recovery itself; for example if a train is delayed, no recovery can make it on time at the next station if there is not enough slack. In this case the local recovery is unrealistic, because disturbances propagate. This weakness of Light Robust models is due to the fact that even though a fist vertical integration is obtained by penalising a function of all local violations in the objective, the recovery actions are still meant to be row wise. Hence, the next step to take is to devise a way to consider global recovery, i.e. a recovery of the solution as a whole.

3.3 Recoverable Robustness

3.3.1 What is it all about: a round up

Robust optimisation is a concept for optimisation under uncertainty. Optimisation under uncertainty seeks for good solutions although the input data, or part of the input data is not known exactly. Thereby, robust optimisation is somehow related to stochastic programming.

Stochastic programming methods, given a probability distribution for the input data, optimise an expected value or an objective including some risk measure. Robust optimisation takes a different approach: a robust solution is a solution that is guaranteed to be feasible for every realisation of the input data except for extreme cases. This has two main advantages. Firstly, the model is easier to solve than most stochastic programming approaches. Therefore, it is the model of choice for large scale instances. Secondly, one can rely on a robust solution in every likely situation, not just on average, but this also gives the main disadvantage of classical robust optimisation.

Stochastic programs usually have at least two stages. In the first stage decisions are taken without knowledge of the data, in the second stage these decisions can be adjusted to the now known exact data. Accordingly, stochastic programming has two types of costs, the deterministic first stage cost and the scenario dependent, i.e., stochastic, second stage cost.

In robust programming there is no second stage decision, nor second stage cost. The first stage decision must be feasible as it is for all considered scenarios. Roughly speaking robust optimisation saves computational effort by not taking a closer look at each scenario, but pays for this in flexibility. This rules out classical robust optimisation for many applications.

Consider the example of a timetable for a railway system. The driving times of the trains are subject to disturbances. Here it is reasonable to restrict to scenarios, in each of which only a few driving times are prolongated, and each of those only to a limited extend. A good timetable will feature some slack. The slack will not be sufficient to make all arrivals on time in every scenario, but the timetable will generally recover from an occasional delay after a few stops. In contrast, a robust timetable must be feasible in every scenario. As a consequence, even if we limit to scenarios where only *one* driving activity is prolongated by at most Δ, the robust timetable must plan Δ prolongated times for *all* driving activities. The standard principle of robustness inhibits the flexibility to tolerate some delayed arrivals. A robust timetabling must insert slack at every driving activity and thus be over-conservative.

In most real-world problems the level of robustness required is far less strict than the one formalised in the classical robust optimisation concept, because solutions may be recovered after the data has changed. In fact, solutions are expected to be feasible for a limited set of scenarios and for a limited recovery in each scenario. Hence, recovery and robustness must be integrated in a new notion.

To overcome this principal weakness of classical robustness, the concept of *Recoverable Robustness* has been introduced. Classical robustness, also called strict robustness, is a special case of this new concept. The concept of recoverable robustness seeks to regain flexibility while remaining aloof from a detailed evaluation of all scenarios. We say, a solution x is *Recoverable Robust* against certain scenarios and for a certain limited type of recovery, if for each of the considered scenarios the solution x can be recovered to a feasible solution within the limits of the type of recovery. Thus, recoverable robustness allows for second stage decisions, the so-called recovery. Yet, the recovery differs in two respects from the second stage decisions of stochastic programming. Firstly, the recoverable robust optimiser need not calculate the recovery cost in each scenario. It suffices to know an upper bound for each scenario, e.g., by calculating the recovery cost for a set of scenarios that will dominate all other scenarios with respect to recovery costs. This will keep the model tractable while providing the desired flexibility. Secondly, the limits to the recovery can be of various kinds, not only cost limits. For example, one may want to impose a limit to the *computational effort* for recovering a timetable. This provides for a broadening of modeling power that is desirable in many real-world applications.

In order to incorporate the afore mentioned modeling power the definition of a recovery robust problem in general entails the specification of a class of admissible algorithms $\mathcal{A}$. We quote the general definition first, and simplify it for our purposes later.

3.3.2 A three step view

The basic idea of Recoverable Robustness [28] is to compute solutions the are robust against a limited set of scenarios and for a limited recovery. In other words, we aim at finding solutions which in a given set of scenarios can be made feasible or recovered by a limited effort.

To do this we need to take three steps:

1. define the *original* optimisation problem (Step O)
2. define the model of imperfect information, i.e. the limited *scenario* set (Step S)
3. define the limited *recovery* possibilities (Step R)

3.3.3 A bit more formal

In the sequel we will give some mathematical details on the definition of Recoverable Robustness, for a complete mathematical treatment refer to Stiller [34]. Let $\mathcal{F}$ denote the original optimisation 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 minimised.

Steps O and S described in 3.3.2 can be carried out using the classical approaches, respectively using integer programming formulations and defining the interval for input parameters. The last step R is new and we need to formalise it by defining a class $\mathcal{A}$ of admissible recovery algorithms.

A solution x for the optimisation problem defined in step O is recovery robust

- against the imperfect information model of step S
- for the recovery possibilities of 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.

Hence, Recoverable Robustness naturally decomposes into two stages: a *planning phase* and a *recovery phase*. The planning phase computes a solution x that may get infeasible in one of the realised scenarios and chooses an algorithm A in $\mathcal{A}$ (one of the admissible recovery algorithms), thus the planning phase defines the pair (x, A). In the recovery phase algorithm A is used to turn x into a feasible solution in the realised scenario. It is important to notice that the recovery phase comes strictly after the planning phase, so the algorithm chosen at the moment of planning cannot be changed during recovery. The given algorithm must be able to recover the solution in every scenario realisation. So the output (x, A) of the planning phase is not only a solution, but it is a *precaution*, in the sense that it does not only give margin for recovery, but explicitly specifies how the recovery will be done.

$\mathcal{S}_\mathcal{F}$ denotes a model of imperfect information for $\mathcal{F}$ in the sense that for every instance O we specify a set $S = S_O \in \mathcal{S}_\mathcal{F}$ of possible scenarios. Let P_s denote the set of feasible solutions in scenario $s \in S$. Note, that $P \cap P_s$ may be empty, and their elements need not even live in the same vector space or represent similar objects in application.

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* $(\mathcal{F}, \mathcal{S}, \mathcal{A})$ *is called a* recovery robust optimisation problem*, abbreviated RROP. For an instance of an RROP we only need to specify the pair* $(O = (P, f), S)$ *since* $\mathcal{A}$ *is identical for all instances. A pair* $(x, A) \in P \times \mathcal{A}$ *is called a* precaution*.*

A precaution is recovery robust*, if for every scenario* $s \in S$*, the recovery algorithm* A *finds a feasible solution to the recovery problem, that is,* $\forall s \in S : A(x, s) \in P_s$*.*

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

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

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.

Note that we can forbid recovery (and thus fall back to strict robustness) by letting $\mathcal{A}$ consist of the single recovery algorithm A with $A(x, s) = x$ for all $s \in S$.

For the remainder we will concentrate on RROPs where the admissible recovery algorithms set a vector of fractional variables y, constrained by a set of linear inequalities, which are defined through the x chosen in the precaution and the scenario s. A vector y is a feasible recovery, if it fulfills those linear inequalities and does not exceed a certain value, the *recovery budget*, in a certain linear cost function, the *recovery cost*. In short, the recovery problem is a linear program defined by the solution x and the scenario s. We make no restrictions to the computational power of the class $\mathcal{A}$. But, as our recovery problem is linear program, we can restrict to polynomial algorithms.

3.3.4 The Price of Robustness and Recoverability

As said, robust optimisation is about trading optimality for feasibility in the scenario set considered. This trade-off is measured in terms of relative loss in the original objective function value [28]. For a recovery-robust optimisation problem $(\mathcal{O}, \mathcal{S}, \mathcal{A})$ denote by $I = (O = (P, f), S)$ a particular instance, where P is the feasible set of the original optimisation problem O, f the corresponding objective function to minimise, S a specific scenario set, $\mathcal{A}$ the set of admissible algorithms, and by $[I]$ the set of all instances.

Definition 2. *The price of robustness of instance I is defined as the ratio:*

$$PoR(I) = \frac{\min_{(x,A)\in P\times\mathcal{A}} \{f(x) \mid \forall s \in S : A(x,s) \in P_s\}}{\min\{f(x) \mid x \in P\}}$$

Definition 3. *The price of robustness for the recovery-robust optimisation problem is:*

$$PoR(\mathcal{O}, \mathcal{S}, \mathcal{A}) = PoR([I]) = \max_{I\in[I]} PoR(I)$$

The recovery algorithm A solves a feasibility problem, but we have not defined any cost related to recovery actions so far. The two most straightforward ways to do this are:

- Put a limit λ_0 on the number (magnitude) of changes λ to the solution x.
- Account for the recovery cost as a new term $g(\lambda)$ in the original objective function, and let the output of the recovery algorithm A also depend on λ. Thus, we require $A(x, s, \lambda) \in P_s$.

Hence, the definition of the recovery-robust problem with recovery costs is:

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

3.3.5 RR for Linear Programs

We start with the case, where also the original problem $\mathcal{F}$ is a linear program. So, we are given a linear program $(\min c'x, \text{ s.t. } Ax \geq b)$. The uncertainty influences the data A and b in a limited way. So in each scenario r the actual data (A^r, b^r) is a limited deviation of the nominal data (A, b) formalized as $(A^r, b^r) \in S := S(A, b)$. Typically we consider the sets $S^\infty(A, b) := \{(A^r, b^r) : \|(A - A^r, b - b^r)\|_\infty \leq \Delta_\infty\}$ or $S^1(A, b) := \{(A^r, b^r) : \|(A - A^r, b - b^r)\|_1 \leq \Delta_1\}$ or their intersection. Hereby, the norms apply to (A, b) as a vector.

The rationale of choosing these sets is that we do not want to design our solution with respect to those extreme scenarios in which either the total deviation from the nominal data, or the local deviation from at least one entry in the nominal data, is beyond a certain threshold.

So for the time being, this restriction of the considered scenario space is motivated on the modeling level. At this point we make no stochastic claim about the relevance of the neglected scenarios, we do not even specify a distribution. Whether this modeling decision is justified, needs to be decided in light of the application and such a decision can entail a lot more factors than the distribution. For example, even if extreme scenarios occur relatively frequent, say in one out of twenty cases, one might still not be willing to plan the business with respect to those cases. One possible reason is that in those cases resources and operating possibilities are activated, which are completely off-beat to the used optimisation model. Often, if customer satisfaction is at stake, very high disturbances of the input are accepted by a majority of the customers as a valid excuse for changes in operation. The customers expect the planning to be able to cope perfectly with a limited number of small disturbances, but they show sympathy for the management of a major disturbance causing inconvenience also to them.

For those scenarios we consider, i.e., for the data realizations in S, we require our solution to be feasible in the sense of recoverable robustness. This means that a solution x need not satisfy $A^r x \geq b^r$ for all $(A^r, b^r) \in S$, but that we can recover the solution with a limited effort. For the linear programming case this 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 feasible in the sense of recoverable robustness, or short, x is recoverable robust, if $\forall (A^r, b^r) \in S : \exists y$ s.t. $A^r x + \hat{A} y \geq b^r$, and $d'y \leq D$.

Thus finding an optimal recoverable robust solution can be formulated as a game by three consecutive players, setting x and (A, b) and y:

$$\min_x c'x \text{ s.t. } D \geq \left\{ \max_{(A,b)\in S} \left\{ \min_y d'y \text{ s.t. } Ax + \hat{A}y \geq b \right\} \right\} \tag{3.1}$$

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

This formalism can also be used to express, if x and y are required to be non-negative, or $x \in P$ for some polytope $P \in \mathbb{R}^n$. But it is a lot more well arranged, if we state such conditions separately:

$$\min_{x\in P} c'x \text{ s.t. } D \geq \left\{ \max_{(A,b)\in S} \left\{ \min_{y\geq 0} d'y \text{ s.t. } Ax + \hat{A}y \geq b \right\} \right\} \tag{3.2}$$

If we fix the strategies of the first two players, i.e., the variables x and (A, b), we get the recovery problem of (3.1): $\min d'y$ s.t. $\hat{A}y \geq b - Ax$, and additionally $y \geq 0$ in the recovery problem of (3.2). The dual of the latter is $\max_{\zeta\geq 0}(b - Ax)'\zeta$ s.t. $\hat{A}'\zeta \leq d$. As we have strong duality for the recovery problem, plugging it into expression (3.2) will not change the problem for the players optimising x or (A, b).

$$\begin{aligned} \min_{x\in P} c'x \text{ s.t. } D &\geq \left\{ \max_{(A,b)\in S} \left\{ \max_{\zeta\geq 0} (b - Ax)'\zeta \text{ s.t. } \hat{A}'\zeta \leq d \right\} \right\} \Leftrightarrow \\ \min_{x\in P} c'x \text{ s.t. } D &\geq \left\{ \max_{(A,b)\in S,\zeta\geq 0} (b - Ax)'\zeta \text{ s.t. } \hat{A}'\zeta \leq d \right\} \end{aligned} \tag{3.3}$$

Consider the maximisation problem in formulation (3.3) for a fixed x, thus find $\max_{(A,b)\in S,\zeta\geq 0}(b - Ax)'\zeta$ s.t. $\hat{A}'\zeta \leq d$. Assume for a moment $\|b - Ax\|_1 \leq \tilde{\Delta}$—an unconventional limit to the scenario space. Now, for each fixed vector ζ the maximum will be attained, if we can set

$\text{sign}(\zeta_i)(b - Ax)_i = \tilde{\Delta}$ for i with $|\zeta_i| = \|\zeta\|_\infty$ and 0 else. In other words, under the previous assumptions $(b - Ax)'\zeta$ attains its maximum when $(b - Ax) = \tilde{\Delta}e_i$ for some suitable $i \in [m]$. Therefore, if we have $\|b - Ax\|_1 \leq \tilde{\Delta}$ as restriction of the scenario space, we can reformulate problem (3.3):

$$\begin{aligned} \min_{x \in P} c'x \text{ s.t. } \forall i \in [m] : D &\geq \left\{ \max_{\zeta \geq 0} (\tilde{\Delta}e_i)'\zeta \text{ s.t. } \hat{A}'\zeta \leq d \right\} \Leftrightarrow \\ \min_{x \in P} c'x \text{ s.t. } \forall i \in [m] : D &\geq \left\{ \min_{y \geq 0} d'y \text{ s.t. } \hat{A}y \geq \tilde{\Delta}e_i \right\} \end{aligned} \tag{3.4}$$

3.3.6 Right-hand side uncertainty

The prima facie awkward condition $\|b - Ax\|_1 \leq \tilde{\Delta}$ is naturally met in those settings where the uncertainty of the data is confined to the right-hand side of the linear program, i.e., $S^p = \{(A^r, b^r) : \|b - b^r\|_p \leq \Delta_p, A^r = A\}, p \in \{1, \infty\}$. Obviously, for S^1 with right-hand side uncertainty, formulation (3.4) is equivalent to a linear program of size $\mathcal{O}(m((n + \hat{n}) \cdot m))$, where $n + \hat{n}$ is the sum of the number columns of A and $\hat{A}$ and m their common number of rows:

$$\min_{x \in P} c'x \text{ s.t. } \forall i \in [m] : D \quad \geq \quad \left\{ \min_{y \geq 0} d'y \text{ s.t. } Ax + \hat{A}y \geq b + \Delta_1 e_i \right\}$$

$$\Leftrightarrow$$

$$\begin{aligned} \min_{x \in P} \quad & c'x \\ \text{s.t.} \quad & \forall i \in [m] : \\ & Ax + \hat{A}y^i \geq b + \Delta_1 e_i \\ & d'y^i - D \geq 0 \\ & x, y^i \geq 0 \end{aligned}$$

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 optimisation 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. We have with the above notation

$$\min_{x \in P'} c(x) \text{ s.t. } D \quad \geq \quad \left\{ \max_{b \in S^1} \left\{ \min_y d'y \text{ s.t. } g(x) + \hat{A}y \geq b \right\} \right\} \tag{3.5}$$

$$\Leftrightarrow$$

$$\begin{aligned} \min_{x \in P'} \quad & c(x) \\ \text{s.t.} \quad & \forall i \in [m] : \\ & g(x) + \hat{A}y^i \geq \Delta_1 e_i + b \\ & d'y^i - D \geq 0 \\ & y^i \geq 0 \end{aligned} \tag{3.6}$$

In particular we can use P' to restrict the variables x_i to integer values. So, assume we are given an integer linear optimisation problem $(\min c'x, Ax \geq b, x \in \mathbb{Z})$ with right-hand side uncertainty. Then we get as its recovery robust version:

$$\min_{x\in\mathbb{Z}} c'x \text{ s.t. } D \quad \geq \quad \left\{\max_{b\in S^1}\left\{\min_y d'y \text{ s.t. } Ax + \hat{A}y \geq b\right\}\right\} \tag{3.7}$$

$$\Leftrightarrow$$

$$\begin{aligned} \min_{x\in\mathbb{Z}} \quad & c'x \\ \text{s.t.} \quad & \forall i \in [m]: \\ & Ax + \hat{A}y^i \geq b + \Delta_1 e_i \\ & d'y^i - D \geq 0 \\ & y^i \geq 0 \end{aligned} \tag{3.8}$$

Defining $f := Ax - b$ we can rewrite (3.7) as follows:

$$\begin{aligned} \min_{f\in\mathbb{Z}\cap\tilde{\mathbb{P}}} \quad & \tilde{c}'f \\ \text{s.t.} \quad & \forall i \in [m]: \\ & f + \hat{A}y^i \geq \Delta_1 e_i \\ & d'y^i - D \geq 0 \\ & y^i \geq 0 \end{aligned} \tag{3.9}$$

with a suitable polytope $\tilde{P}$ and a corresponding linear objective $\tilde{c}$. In other words, for solving the recovery robust version, the original, deterministic, linear, integer optimisation problem can be left untouched. We only have to stick 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 to robustness is evaluated.

We arrive at the following formulation for a recovery robust integer linear optimisation problem $(\min c'x, Ax \geq b, x \in \mathbb{Z})$ with a linear programming recovery and right-hand side uncertainty limited by Δ_1 in the one norm.

$$\begin{aligned} \min_{f\in\mathbb{Z}} \quad & c'f \\ \text{s.t.} \quad & Ax - b = f \\ & \forall i \in [m]: \\ & f + \hat{A}y^i \geq \Delta_1 e_i \\ & d'y^i - D \geq 0 \\ & f, y^i \geq 0 \end{aligned} \tag{3.10}$$

3.3.7 Robust Network Buffering

Next we concentrate on a special kind of recovery, which at first glance does not seem to be a recovery at all.

Consider again the problem of a railway timetable, where the driving activities are subject to occasional prolongation. The task is to construct a timetable for n events, arrivals and departures, i.e., to define an n-dimensional vector x, fixing the points in time when the n events are scheduled to take place. The feasibility of the timetable is determined by a set of precedence constraints that can be expressed in a directed (acyclic) graph $G(V, E)$ with n vertices and a function $t : E(G) \to \mathbb{R}$. Each vertex corresponds to one of the events, and an arc e from event a to event b symbolizes that b must not take place earlier than $x_a + t(e)$. A simple variant of the model features driving arcs, stopping arcs and transfer arcs. A driving arc $e = (a, b)$ leads from a departure a of a train at a station to the arrival b of the same train at its next station. The time $t(e)$ is the technical travel time of the train on that track. A stopping arc represents the stop of a train in a station, and its time, $t(e)$, the minimal time the train has to stop at the station to let passengers get off and on. A transfer arc $e = (a, b)$ expresses that a significant amount of passengers wishes to transfer from the train of the arrival event a at a certain station to the other train, of which b is the departure event at the same station. Therefore $t(e)$ of a transfer arc e is the time passengers need to change from one train to the other.

By virtue of the application we can assume t to be non-negative. Then G being acyclic is a trivial prerequisite for the existence of a feasible solution, which is also sufficient. We also define a weight function $w : E(G) \to \mathbb{R}$. The weight $w(e)$ of an arc e in this setting, is the number of passengers using for their trip the transfer, driving, or stopping activity represented by e.

The recovery robust version of this problem is the following:

$$\begin{aligned} \min_{x,f\geq 0} \quad & \textstyle\sum_{e=(a,b)\in E} w(e)(x_b - x_a) \\ \text{s.t.} \quad & x_b - x_a - t(e) = f_e, \forall e = (a, b) \in E \\ & D \geq \left\{ \max_{b\in S^1} \left\{ \min_{y\geq 0} d'y, \text{s.t. } f_e + y_b - y_a \geq b, \forall e = (a, b) \in E \right\} \right\} \end{aligned} \tag{3.11}$$

Here the recovery cost d_a of an event expresses the importance of a delay at this event. The set $S^1 := \{b \geq 0 : \|b\|_1 \leq \Delta_1\}$ is defined as in the previous sections. Then, by the previous line of argument, this problem (3.11) is equivalent to:

$$\begin{aligned} \min_{x} \quad & \textstyle\sum_{e=(a,b)\in E} w(e)(x_b - x_a) \\ \text{s.t.} \quad & x_b - x_a - t(e) = f_e, \qquad \forall e = (a, b) \in E \\ \forall i \in E : \quad & \\ & f_e + y^i_b - y^i_a \geq \Delta_1 \xi_e(i), \quad \forall e = (a, b) \in E \\ & d'y^i - D \geq 0 \\ & f, y^i \geq 0 \end{aligned} \tag{3.12}$$

where ξ is the characteristic function on E, i.e., equal to 1 if i equals e and 0 else. The recovery variable y^i_a expresses the delay of an event a in the scenario where edge i is delayed by Δ_1 time units. Notice that variables f_e represent the slacks or *buffers* over the arcs of the delay propagation network defined by G.

This is a typical situation: we have an optimisation problem on a network. The result of that will have to be operated under disturbances. The disturbances propagate through the network in a way depending on the solution of the optimisation problem. The solution of the original optimisation 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 optimisation will construct the cheapest x vector to ensure that buffering. We want to formulate this in a more general way.

Given a optimisation 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 by a bound in the 1-norm, we get the following: the recovery robust version of P, in which the cost of propagated disturbance must be kept below a fixed budget D, can be formulated as the original problem P plus a linear program.

What we get is a general method to buffer networks. In the exact model the actions of planer, adversary, and recovery are intertwined. By virtue of the above analysis we can keep any efficient formulation for the non-robust optimisation problem unchanged, and stick a linear program to it, in order to correctly model the recoverable robustness. For example, we can use the highly developed techniques for timetabling in MIP formulations and specialized solving procedures, and add exteriorly the linear program for the network buffering.

3.3.8 Summary

Let (P, ϕ) be a well understood optimisation problem, hard in principal, but solvable for considerable and practically relevant size instances, like many integer programs. Now, we want to solve a robust version of it. A strict robust approach may not be suitable, because it is over-conservative. In case, we can interpret the influence of the uncertain data, as a disturbance propagating along some directed graph $G(V, E)$; and if we can efficiently interpret any solution $x \in P$ as a buffering against this propagation, than we can supplement the powerful solving methods for (P, ϕ) by a linear program to get a recovery robust solution.

$$\min_{(x,f)\in P'} \phi(x) \text{ s.t. } D \quad \geq \quad \left\{ \max_{b\in S^1} \left\{ \min_{y\geq 0} d'y \text{ s.t. } y_b - y_a + f_e \geq b, \forall e \in E \right\} \right\}$$

$$\Leftrightarrow$$

$$\begin{aligned} & \min_{(x,f)\in P'} \phi(x) \\ \text{s.t. } \forall e \in E: & \\ & y_b^e - y_a^e + f_i \quad \geq \Delta_1 \xi_e(i),\ \forall i = (a,b) \\ & d'y_i^e - D \quad \geq 0 \\ y \geq 0 & \end{aligned}$$

where P' is a subspace of the product space $P \times \mathbb{R}^m$, and $S^1 := \{b \geq 0 : \|b\|_1 \leq \Delta_1\}$.

Chapter 4

Price of Recoverability for Railway Rolling Stock Planning

In this chapter we explore the possibility of applying the notions of Recoverable Robustness and Price of Recoverability, introduced by [28], to railway rolling stock planning, being interested in recoverability measures that can be computed in practice, thereby evaluating the robustness of rolling stock schedules. In order to lower bound the Price of Recoverability for any set of recovery algorithms, we consider an "optimal" recovery algorithm and propose a Benders decomposition approach to assess the Price of Recoverability for this "optimal" algorithm. We evaluate the approach on real-life rolling stock planning problems of NS, the main operator of passenger trains in the Netherlands. The preliminary results show that, thanks to Benders decomposition, our lower bound can be computed within relatively short time for our case study.

4.1 Introduction

Recently [28] introduced the concept of Recoverable Robustness as a generic framework for modelling robustness issues in railway scheduling problems. Also, the Price of Recoverability (PoR) was defined as a measure of recoverability. However, this notion is mainly of theoretical nature, and cannot be used in a straightforward way for concrete problems. In particular, computing PoR requires, according to [28], minimisation over a set of recovery algorithms, and it is not clear how this can be carried out.

Reference [16] considers the robustness of shunting problems. It overcomes these difficulties by analysing PoR for a few *concrete* recovery algorithms and by proving lower and upper bounds.

The purpose of this chapter is to investigate another way of bringing the theoretical notion of PoR closer to practice. In particular, we consider what happens if recovery is done in the best possible way: the resulting value of PoR for this unique "optimal" recovery algorithm is then a lower bound on the value of PoR for *any* set of recovery algorithms. Under mild assumptions, this lower bound can be computed by solving a single mathematical program.

We address the practical evaluation of the lower bound above for a specific case study, concerning the medium-term rolling stock planning problem of NS, the main operator of passenger trains in the Netherlands. It arises 2–6 months before the actual train operations, and amounts to assigning the available rolling stock to the trips in a given timetable. The

objectives of the problem that is traditionally solved, call nominal problem, are related to service quality, efficiency, and – to a limited extent – to robustness. Reference [20] describes a Mixed Integer Linear Programming (MILP) model for this nominal problem. Using commercial MILP software, the solution times on real-life problems of NS are quite low, ranging from a few minutes (on most instances) to a couple of hours (on some particularly complex instances). A software tool based on this model has been in operation within NS since 2004.

The solutions of the nominal problem are optimal under undisrupted circumstances only. However, infrastructure failures, bad weather, and engine break-downs often lead to disruptions where the nominal solution cannot be carried out any more. In such cases, disruption management must take place to come up with an adjusted rolling stock schedule. In this re-scheduling process, the original objective criteria are of marginal importance, the goal being to quickly find a feasible solution that is "close" to the nominal one and can be implemented in practice.

The computation of the lower bound on PoR mentioned above for our case study requires the solution of a very-large Linear Programming (LP) model, in which several possible disruption scenarios are considered. We propose a Benders decomposition approach for the solution of this LP, leading to a subproblem for each scenario. Our preliminary computational results on a real-life rolling stock planning instance of NS indicate that the method takes relatively short time, and widely outperforms the straightforward solution of the whole LP by a state-of-the-art solver.

This chapter is structured as follows. In Section 4.2 we quote the definition of the Recoverable Robustness and Price of Recoverability from [28]. In Section 4.3 we describe the lower bound that we consider, based on a "best possible recovery" policy, and the associated mathematical programming problem. Section 4.4 describes the railway rolling stock scheduling problem of NS. Section 4.5 is devoted to our preliminary computational results. Finally, Section 4.6 outlines our plans for further research.

4.2 The Price of Recoverability

In this section we give a short summary of the definition of the Price of Recoverability by [28]. The main idea is to compute a solution to an optimisation problem and at the same time to analyse the recovery costs in case of disturbed input data. More concretely, one considers a (limited) set of scenarios with their own feasible regions, as well as a set of admissible recovery algorithms. The objective is to find a solution of the original (nominal) problem *and* a recovery algorithm in the given set. The requirement is that, using the recovery algorithm, the solution of the original optimisation problem can be transformed to a feasible solution of each scenario at "low cost". The Price of Recoverability measures both the objective function of the original problem and the recovery costs.

The set of admissible recovery algorithms can be chosen in several ways. One may consider algorithms with limited (e.g. linear) running time, or algorithms that are obtained from a particular heuristic framework (e.g. a crew re-scheduling algorithm based on iterated crew duty swaps).

The notions of Recoverable Robustness and Price of Recoverability are defined formally as follows. First of all, we are given a *Nominal Problem* of the form:

$$\mathrm{NP} = \min\{c(x) \mid x \in K\}, \tag{4.1}$$

where $x \in \mathbb{R}^n$ is the variable vector, $K \subseteq \mathbb{R}^n$ is the feasible region and $c : K \to \mathbb{R}_+$ is the cost function.

Moreover, we are given a set $\mathcal{S}$ of *scenarios*; each scenario $s \in \mathcal{S}$ having its own feasible region K_s. (For example, a scenario may refer to the case of cancelling some trains due to infrastructure failure, thereby requiring some kind of recovery action.) Furthermore, we are given a set $\mathcal{A}$ of *recovery algorithms*: a recovery algorithm $A \in \mathcal{A}$ takes on input a nominal solution $x \in K$ and a scenario $s \in \mathcal{S}$ and produces a solution $A(x, s) \in K_s$ which is feasible in scenario s. Finally, we are given, for $s \in \mathcal{S}$, a function $d_s : K \times K_s \to \mathbb{R}_+$ measuring the *deviation* $d_s(x, x_s)$ of a solution x_s for scenario s from a nominal solution x, and a monotone non-decreasing function $f : \mathbb{R}_+^{\mathcal{S}} \to \mathbb{R}_+$ penalising the deviation over all scenarios.

The *Recovery-Robust Optimisation Problem* defined in [28] is then:

$$\mathrm{RROP}_{\mathcal{A}} = \min\{c(x) + f(z) \mid x \in K,\ A \in \mathcal{A},\ z_s = d_s(x, A(x, s))\ \ (s \in \mathcal{S})\}, \tag{4.2}$$

where $z = (z_{s_1}, z_{s_2}, \ldots) \in \mathbb{R}_+^{\mathcal{S}}$ is a vector of auxiliary variables representing the deviations.

Reference [28] chooses $f(z) = \max_{s \in \mathcal{S}} z_s$, i.e. penalises the *maximum deviation* in the objective function. A stochastic-programming approach would be to define a probability p_s for each scenario $s \in \mathcal{S}$, and to consider $f(z) = \sum_{s \in \mathcal{S}} p_s z_s$, penalising the *expected (average) deviation*.

The Price of Recoverability (PoR) is then defined as the ratio between the optimal values of the Recovery-Robust and the Nominal Problem:

$$\mathrm{PoR}_{\mathcal{A}} = \frac{\mathrm{RROP}_{\mathcal{A}}}{\mathrm{NP}}. \tag{4.3}$$

Reference [28] also compares Recoverable Robustness to well-known concepts such as stochastic programming (see e.g. [7]) or robust optimisation ([5]), and discusses the similarities and differences of the approaches to capture robustness.

4.2.1 Reformulation of PoR

Define the function

$$\Phi_A(x) = c(x) + f\left(d_{s_1}(x, A(x, s_1)), d_{s_2}(x, A(x, s_2)), \ldots\right).$$

Then $\mathrm{RROP}_{\mathcal{A}}$ corresponds to the minimisation of the function Φ:

$$\mathrm{RROP}_{\mathcal{A}} = \min_{A \in \mathcal{A}} \min_{x \in K} \Phi_A(x). \tag{4.4}$$

In later sections of this chapter we shall consider a simplified version for the case in which $\mathcal{A}$ contains a single algorithm A only:

$$\mathrm{RROP}_{\{A\}} = \min_{x \in K} \Phi_A(x). \tag{4.5}$$

4.2.2 How to Compute PoR?

The definition (4.3) (via the definition (4.2)) requires minimisation over the set $\mathcal{A}$ of recovery algorithms. How (and if) this can be done clearly depends on how the set $\mathcal{A}$ is specified. In any case, one can follow (at least) two approaches to compute (or approximate) PoR.

In the first approach, one considers a class of small and well-behaved problems together with a small set of recovery algorithms. Then one proves worst-case bounds on PoR by an appropriate theoretical analysis. Reference [16] reports such results for the shunting problem. With our notation, such an approach essentially amounts to deriving bounds on the minimum of the function Φ_A for each $A \in \mathcal{A}$. Note that this approach is likely to succeed on fairly simplified test problems; real-life (railway) scheduling problems often have features that cannot be handled easily in theoretical worst-case proofs.

In this chapter we follow a second approach, namely we restrict attention to the best possible recovery algorithm, observe that the computation of PoR for this single algorithm leads to a lower bound on PoR for each set $\mathcal{A}$, and numerically solve a mathematical programming problem to compute the value of this lower bound for a particular real-life railway resource scheduling problem. Therefore the results that we obtain are of empirical nature.

4.3 PoR with an Optimal Recovery Algorithm

Let $\mathcal{A}_{\text{all}}$ be the set of *all* recovery algorithms and define

$$A_{\text{opt}}(x, s) = \arg\min\{d_s(x, x_s) \mid x_s \in K_s\}. \tag{4.6}$$

In words, for each scenario s and for each $x \in K$, A_{opt} determines the solution in K_s with the smallest possible deviation from x. That is, A_{opt} represents the best possible recovery action. This is formalised in the following proposition.

Proposition 1. $\text{RROP}_{\mathcal{A}_{\text{all}}} = \text{RROP}_{\{\text{A}_{\text{opt}}\}}$.

Proof. Clearly, $\text{RROP}_{\mathcal{A}_{\text{all}}} \leq \text{RROP}_{\{\text{A}_{\text{opt}}\}}$. On the other hand, for each $x \in K$, $A \in \mathcal{A}$ and $s \in \mathcal{S}$ we have

$$d_s(x, A(x, s)) \geq d_s(x, A_{\text{opt}}(x, s))\,.$$

Therefore

$$\min_{A \in \mathcal{A}_{\text{all}}} \Phi_A(x) \geq \Phi_{A_{\text{opt}}}(x)$$

and (4.4) yields

$$\text{RROP}_{\mathcal{A}_{\text{all}}} \geq \text{RROP}_{\{\text{A}_{\text{opt}}\}}.$$

□

In other words, the minimum of (4.2) if $\mathcal{A} = \mathcal{A}_{\text{all}}$ is attained at A_{opt}. This of course implies that the minimum of (4.2) for a generic $\mathcal{A}$ cannot be better than $\text{RROP}_{\{\text{A}_{\text{opt}}\}}$, as stated in the following corollary.

Corollary 2. $\text{RROP}_{\mathcal{A}} \geq \text{RROP}_{\{\text{A}_{\text{opt}}\}}$ *for every set* $\mathcal{A}$ *of algorithms.*

This implies that the computation of $\text{RROP}_{\{\text{A}_{\text{opt}}\}}$ yields a lower bound on $\text{RROP}_{\mathcal{A}}$, and therefore $\text{PoR}_{\text{A}_{\text{opt}}}$ a lower bound on $\text{PoR}_{\mathcal{A}}$, for a generic set of recovery algorithms $\mathcal{A}$. Moreover,

$$\text{RROP}_{\{A_{\text{opt}}\}} = \min\{c(x) + f(z) \mid x \in K,\ x_s \in K_s\ (s \in \mathcal{S}),\ z_s = d_s(x, x_s)\ \ (s \in \mathcal{S})\}. \tag{4.7}$$

That is, $\text{RROP}_{\{\text{A}_{\text{opt}}\}}$ is the optimum value of a mathematical program, which is not the case for $\text{RROP}_{\mathcal{A}}$ for a generic $\mathcal{A}$. This is the reason why in this chapter we focus our attention on the practical computation of the former.

4.3.1 Solution Methodology

For the sake of concreteness, we will restrict our attention to the case of $\mathrm{RROP}_{\{\mathrm{A_{opt}}\}}$ in which the following hold:

- $f(z) = \max_{s\in\mathcal{S}} z_s$, i.e. only the largest deviation is penalised in the objective function;
- $c(x) = c^\mathsf{T} x$ for a given $c \in \mathbb{R}^n$, i.e. the objective function is linear;
- $x \in K$ can be expressed as $Ax \geq b$ for given $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, i.e. feasibility of a nominal solution can be expressed by linear constraints (and possibly by the integrality of some components of x, see below);
- for each $S \in \mathcal{S}$, $x_s \in K_s$ can be expressed as $A_s x_s \geq b_s$ for given $A_s \in \mathbb{R}^{m_s\times n}$ and $b_s \in \mathbb{R}^{m_s}$;
- for each $S \in \mathcal{S}$, $z_s = d_s(x, x_s)$ can be expressed as $z_s = d_s^\mathsf{T} x + e_s^\mathsf{T} x_s + g_s$ for given $d_s, e_s \in \mathbb{R}^n$ and $g_s \in \mathbb{R}$, i.e. the deviation is a linear function of the nominal and the recovered solution.

If we include the possible integrality restriction on some of the x and x_s components, the above assumptions are not really restrictive, since they amount to require that the nominal problem, the feasibility of a recovered solution, and the value of the deviation can be expressed as a MILP. In the computational experiments carried over in this chapter, we will restrict attention to the case in which such integrality restriction is not imposed. Depending on the specific application, this may be the case, or it may lead to solution of the LP relaxation of the actual MILP, which yields a lower bound on $\mathrm{RROP}_{\{\mathrm{A_{opt}}\}}$ and therefore on $\mathrm{RROP}_{\mathcal{A}}$ for each $\mathcal{A}$. In any case, the Benders decomposition approach that we illustrate can easily be modified to have integrality restrictions on the x variables. Since the purpose of this chapter is to study the possibility to practically compute lower bounds on PoR for real-world instances, it is natural to restrict attention to LP relaxations.

Given the above assumptions, (4.7) can be formulated as follows, where λ is an auxiliary variable expressing the deviation penalty:

$$\begin{aligned}
\min \quad & c^\mathsf{T} x & & + \lambda & & & (4.8)\\
\text{s.t.} \quad & Ax & & & \geq b, & & (4.9)\\
& & A_s x_s & & \geq b_s, & \quad \forall s \in \mathcal{S}, & (4.10)\\
& - d_s^\mathsf{T} x - & e_s^\mathsf{T} x_s & + \lambda & \geq g_s, & \quad \forall s \in \mathcal{S}. & (4.11)
\end{aligned}$$

For solving (4.8) – (4.11) one can apply various mathematical programming techniques. In this chapter we focus on Benders decomposition (also known as L-shaped method) (see e.g. [31]), a cutting plane method that exploits the block-diagonal structure of the problem. This is an approach widely used for such problems (such as for stochastic programming).

Briefly, the Benders decomposition approach keeps solving the (gradually extended) nominal problem (4.8) – (4.9). Based on the current optimal solution, the feasibility of the subproblem (4.10) – (4.11) is checked. The procedure terminates if the subproblem is feasible, in which case the current optimal solution is optimal also for (4.8) – (4.11). In case of infeasibility, inequalities in terms of x and λ are derived and added to the nominal problem, and the updated nominal problem is re-optimised.

Benders decomposition is applicable if the subproblems are LPs, i.e. if the x_s variables are continuous, whereas integrality on the x variables can be handled, although (as already mentioned) it will be relaxed in our computational experiments.

4.4 The Test Problem: Rolling Stock Re-scheduling

This section is devoted to the description of the specific real-world case study on which we focused our attention, for a complete treatment refer to Maróti [30]. We considered the Rolling Stock planning problem at *NS Reizigers* or NSR (NS Passenger). NS is the largest passenger operator in the Netherlands and is divided into several parts. NSR, in particular, is responsible for operating the trains: it cares for timetable plans, rolling stock and crew schedules and is also responsible for carrying out the plans themselves.

4.4.1 The Planning Process

It is well known that the railway planning process is divided into several steps, and there are many ways to classify them. There are three main common criteria: one is based on the length of the planning horizon, the other on the target and finally on the geography. The first distinguishes:

- *Strategic planning*: has a planning horizon of several years and concentrates on capacity planning.
- *Tactical planning*: considers capacity planning issues with 2 months up to a year horizon.
- *Operational planning*: it is meant to adjust tactical plans for the forthcoming weeks.
- *Short-term planning*: deals with the problems that arise when a train is operated or a few days before that.

The second considers:

- *Timetabling*: provides a timetable for a number of trains on a certain part of the railway network.
- *Rolling Stock scheduling*: defines the required capacity of rolling stock for passenger transportation.
- *Crew Scheduling*: builds the work schedules of crews in order to cover a planned timetable.

The last considers two types of planning steps: *central* and *local*. Central planning affects the entire railway network, whereas local planning impacts only on a single station.

4.4.2 Tactical Rolling Stock Planning at NSR

Given a timetable plan, the rolling stock has to be scheduled in order to carry out timetable services by meeting the required capacity for passenger transportation. Rolling stock schedules consist of two parts: the centrally planned *tactical rolling stock circulations*, that assign the available rolling stock to the timetable services, and the locally created *tactical shunting*

plans, that specify the train movements inside the station. Our case-study deals with a specific step of the former. In fact, the first step of rolling stock planning assigns the available rolling stock to the so called *line groups*. A line group is a collection of interconnected lines. The reason for this decomposition is twofold. First, the rolling stock scheduling problem would be way too large to be solved. Second, the decomposition hedges against disturbances, hence improves on the robustness of the plan. In fact, if trains in a particular line group are affected by disturbances, the delays will not spread to other line groups directly. Of course, train movements belonging to different line groups may conflict inside the station, but the local planner will care for that. The next step, the one we consider, determines the rolling stock circulation for every line group. The schedules apply for a day, and the task of the planner is to make sure that consecutive train schedules can be attached to each other by considering the number of train units that are left during the night in a station and those available the next morning. Central planners have little or no information on the shunting possibilities at each station, therefore local planners receive a *draft* rolling stock schedule and communicate their feedback to the central planners in case the proposed plan leads to unsolvable shunting problems. As we will see later, the main criteria of rolling stock scheduling are efficiency, service quality and reliability, that reflect into these goals: reducing carriage-kilometres, provide enough seat capacity, and keep the number of composition changes as low as possible.

4.4.3 Locomotives vs Multiple units

Here we present some technical specific features of rolling stock at NSR. A *locomotive* is a railway vehicle that provides the motive power for a train. A locomotive has no payload capacity of its own, and its sole purpose is to move the train along the tracks. Traditionally, locomotives pull trains from the front. Increasingly common is push-pull operation, where a locomotive pulls the train in one direction and pushes it in the other, and is optionally controlled from a control cab at the opposite end of the train. In contrast, some trains have self-propelled vehicles. These are not normally considered locomotives, and may be referred to as *multiple units*. The use of these self-propelled vehicles is increasingly common for passenger trains, but very rare for freight. NSR mainly uses units instead of locomotive-hauled carriages. The reason is that by having a driver's seat on both ends of unit, the turnaround process in case of direction changes can be carried out easily and quickly. Moreover, the shunting process for self-propelled units is easier than for locomotive-hauled carriages. There are different types of units according to the number of carriages they are composed of, and these can be combined forming compositions of different lengths able to meet passenger demands for the various train services.

4.4.4 Successors and predecessors

A unit that arrives at a station as part of a train can continue its daily duty in one of the several trains departing from that station or may be left in the station on a specific shunting area or depot. Arriving units usually go over to the earliest departing train that belong to the same line group and needs a unit of its type. We call it *successor train* of the arriving train. The arriving train itself is the *predecessor* of its successor train. In a station it may be possible to adjust the composition of a train. Thus, units can be *uncoupled* and placed in a shunting yard and other units, that have been stored at the station, may be added (*coupled*) to the train before departure. The layout of the station determines what kind of composition

Figure 4.1: Steam locomotive as operated by the Victorian Railways of Australia.

Figure 4.2: A classic Belgian multiple unit.

changes are possible.

4.4.5 The Shunting Process

Before showing the model in detail, it is necessary to give a quick overview on the shunting process and the corresponding simplified model used at NS for tactical and operational planning. The basic assumption in this model is that time is the most binding bottleneck in the shunting process, in other words any shunting operation, however complicated, can be carried out in any shunting yard if the crew has enough time for it. For this purpose it is important to introduce the concept of *inventory*, which is the number of units per type that are stored at a given moment at a given station, these units can be added to any trip departing from there. Consider an arriving trip t and its successor trip t'. A composition change between trip t and t' means that while most units of t go over to t', others may be coupled or uncoupled. Notice that in this simplified model only coupled or uncoupled units increase and decrease the inventory. Whereas units that travel through a station with a short stop are not part of the inventory. Uncoupled units can be used later for other trips. In order to give enough time for the necessary shunting operations, uncoupled units do not get available, by increasing the inventory, immediately (i.e. as soon as the trip arrives at the station), but after a certain *reallocation time*, generally 30 minutes, which is constant and independent from the particular shunting operation. Figure 4.3 shows two trips, t and its successor t' (arrows) arriving and departing from a given station, represented by a double time-line. The upper line shows the arrival and departure times of the trips, the lower one shows when the corresponding changes

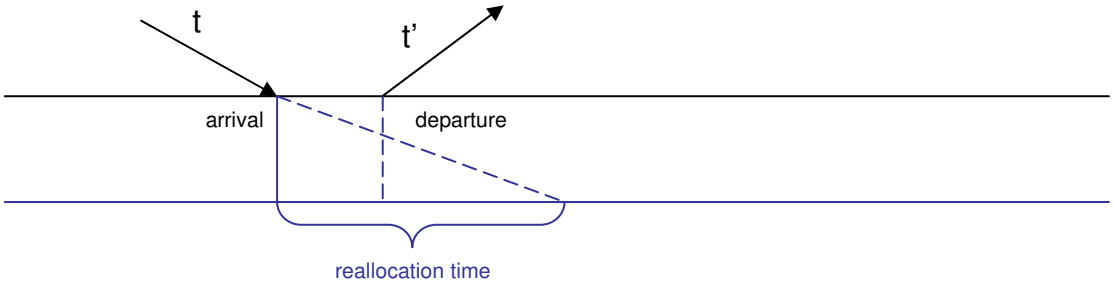

Figure 4.3: Reallocation time (a).

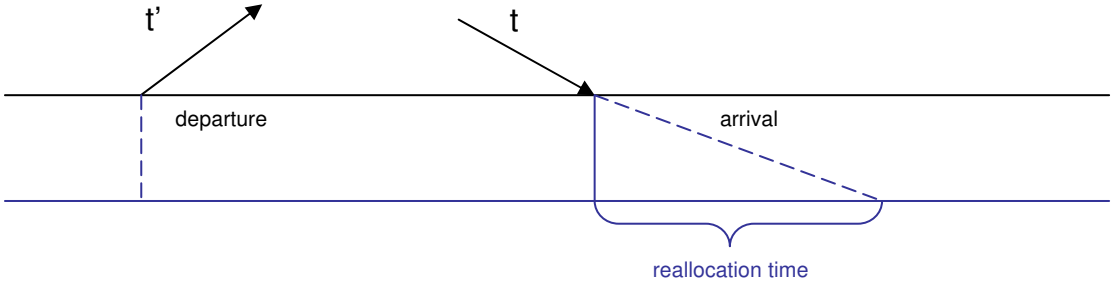

Figure 4.4: Reallocation time (b).

in the inventory take place: for arriving trip t, units are detached and added to the inventory after the reallocation time, whereas units of departing trips are taken from the inventory. Similarly, a trip t that departs from a station and has no predecessor gets all its units from the inventory of the station. Units of trips that arrive at the station and have no successor go to the inventory of the arrival station. As before, the reallocation time must be met when uncoupled units increase the inventory, as shown in picture 4.4

4.4.6 The Nominal Problem: an overview

We consider the medium-term railway rolling stock scheduling problem of NS. It arises 2–6 months before the actual railway operations, and has the task of assigning the available rolling stock to the *trips* in a given timetable. In this section we give a brief problem description. Further details about the problem can be found in [20] and in [30].

The rolling stock consists of *units*. Each unit has driver's seats at both ends and an own engine. It is composed of a number of carriages, and cannot be split up in every-day

operations. Units are available in different *types* and can be combined with each other to form *compositions*. This allows a fine adjustment of the seat capacity to the passenger demand.

The timetable of NS is quite dense, and the turning time of the trains is short, often less then 20 minutes. The rolling stock connections are explicitly given in the input timetable by the *successor trips*: The units that serve in a trip go over to the successor trip, even though certain *composition changes* can take place. Due to the short turning times, the composition change possibilities are limited to *coupling* or *uncoupling* of one or two units at the appropriate side of the train.

The objective is three-fold. *Service quality* is measured by seat shortage kilometres. It is computed by comparing the assigned seat capacity to the a priori given expected number of passengers; by multiplying the number of unseated passengers by the length of the trip; and finally by summing these values over all trips. *Efficiency* is expressed by the carriage-kilometres which is roughly proportional both to the electricity or fuel consumption and to the maintenance costs. *Robustness* is taken into account by counting the number of composition changes. Indeed, coupling or uncoupling of units causes additional traffic through the railway nodes, and thereby may lead to delay propagation if some passing trains are late.

We note again that the the nominal problem is solved several months before the operations. This leaves enough time to plan the low-level train operations at the railway nodes. In particular, shunting drivers are scheduled to carry out the coupling and uncoupling operations. Moreover, the end-of-day rolling stock balances are such that the units are at the right place for the next day's operations.

In order to define a MILP for the problem, the set of rolling stock types is denoted by M, the set of trips by T, the set of compositions by P, and the set of stations by S. For any $m \in M$, a_m denotes the number of available rolling stock units of type m.

The main binary decision variables are $x_{t,p}$, expressing whether composition p is assigned to trip t. Moreover, we have the binary variables $z_{t,p,p'}$ whose value is 1 if trip t has composition p and if the successor of t has composition p'. The z variables are only defined for those triples (t, p, p') where the composition change from p to p' is allowed after trip t, i.e. the constraints on the composition changes are implicitly represented by these variables.

The stations are modelled by the *inventories*. The inventory of a station at a certain time instant consist of all units that are located there. The basic rule is that units to be coupled to a train are pulled from the inventory immediately upon departure, while uncoupled units are added to the inventory a certain time (say 30 minutes) after arrival. This ensures enough time for necessary shunting operations.

The integer variables $y_{t,m}$ count the inventories of the units of type m at the departure station of trip t right after the departure of trip t. The beginning-of-day and end-of-day inventories of station s of type m are represented by the variables $y^0_{s,m}$ and $y^\infty_{s,m}$.

Letting the successor of trip t be denoted by $\sigma(t)$, the departure station of t be denoted by $d(t)$, c, d be appropriate objective function coefficients, and α, β, γ be appropriate inventory

coefficients, a MILP formulation is the following.

$$\min \sum_{t\in T}\sum_{p\in P} c_{t,p}x_{t,p} + \sum_{t\in T}\sum_{p\in P}\sum_{p'\in P} d_{t,p,p'}z_{t,p,p'} \tag{4.12}$$

$$\text{s.t.} \sum_{p\in P} x_{t,p} = 1, \qquad \forall t \in T, \tag{4.13}$$

$$x_{t,p} = \sum_{p'\in P} z_{t,p,p'}, \qquad \forall t \in T, p \in P, \tag{4.14}$$

$$x_{\sigma(t),p'} = \sum_{p\in P} z_{t,p,p'}, \qquad \forall t \in T, p' \in P, \tag{4.15}$$

$$\begin{aligned} y_{t,m} = y^0_{d(t),m} &+ \sum_{t'\in T}\sum_{p\in P}\sum_{p'\in P} \alpha_{t,t',p,p',m} z_{t',p,p'} \\ &+ \sum_{t'\in T}\sum_{p\in P} \beta_{t,t',p,m} x_{t',p}, \qquad \forall t \in T, m \in M, \end{aligned} \tag{4.16}$$

$$y^\infty_{t,m} = y^0_{d(t),m} + \sum_{t'\in T}\sum_{p\in P}\sum_{p'\in P} \gamma_{t,t',p,m} x_{t',p}, \quad \forall t \in T, m \in M, \tag{4.17}$$

$$\sum_{s\in S} y^0_{s,m} = a_m, \qquad \forall m \in M, \tag{4.18}$$

$$x_{t,p}, z_{t,p,p'} \text{ binary}, \qquad \forall t \in T, p \in P, p' \in P, \tag{4.19}$$

$$y_{t,m}, y^0_{s,m}, y^\infty_{s,m} \geq 0, \text{ integer}, \qquad \forall t \in T, s \in S, m \in M. \tag{4.20}$$

The objective (4.12) takes into account the trip assignments and the compositions of consecutive trips. Constraints (4.13) state that each trip gets exactly one composition. Constraints (4.14) and (4.15) link the z variables to the x variables. Constraints (4.16) and (4.17) compute the inventories with appropriate coefficients α, β and γ. Constraints (4.18) specify the available rolling stock.

The objective function can incorporate a wide variety of objective criteria related to service quality, efficiency and robustness. Experience shows that, for the practically meaningful objective coefficients, the LP relaxation of the model above is very tight, the associated lower bound being always within a few percents of the MILP optimum. The MILP model can be solved for medium-sized instances of NS within a few seconds to optimality. Simple LP rounding heuristics turned out to be powerful for the most challenging problem instances.

4.4.7 The Scenarios and the Associated Deviations

In our robustness framework, the solutions of the nominal problem are to be operated subject to disruption scenarios. Each scenario is obtained by assuming that a certain part of the network is blocked for a certain time interval of several hours. All the trips that interfere with the infrastructure blockage are removed. Such disruptions are quite common in practice. These are the ones that require significant resource re-scheduling.

It is worthwhile to note that the timetabling and resource scheduling decisions are strictly separated. In the Netherlands, for example, an independent infrastructure managing authority is responsible for the timetable adjustments, while the railway operators themselves are responsible for resource re-scheduling. Therefore from the resource planning's point of view, the adjusted timetable that takes care of the disruption is to be considered as input.

We assume that a disruption becomes known at the beginning of the blockage. The task is then to re-schedule the rolling stock from that point on till the end of the day. The solution has to fulfil the same requirements as the nominal problem, the only additional option being to cancel a trip.

In this research we also assume that the exact duration of the disruption is known at its beginning. Admittedly, this assumption is very optimistic for practical purposes. On the other hand, it simplifies the mathematical model, and still enables one to gain insight of the recovery capacity of rolling stock schedules.

The three main criteria in re-scheduling are as follows (in decreasing order of importance): (*i*) minimise the number of cancelled trips; (*ii*) minimise the number of newly introduced couplings and uncouplings; (*iii*) minimise the deviation of the planned end-of-day rolling stock balance. The first criterion limits the passenger inconvenience. The second criterion aims at keeping the schedule of the shunting drivers intact. The third criterion tries to restrict the consequences of the disruption on a single day.

Although the model (4.12) – (4.20) was originally developed for the nominal problem, it can be adjusted for rescheduling as well. That is, the feasibility of a recovered solution and the associated recovery costs can be computed as a variant of the model above. We express the model for a single scenario, omitting the index s that represents the scenario and noting that here s stands for the index of a station.

First of all, constraints (4.13) – (4.20) with variables $\widetilde{x}$, $\widetilde{z}$, $\widetilde{y}^0$ and $\widetilde{y}^\infty$ are to be stated for the trips of each scenario. In this case, as anticipated, we also allow the empty composition $\emptyset$, where $\widetilde{x}_{t,\emptyset} = 1$ means that trip t is cancelled. Then, one has to impose constraints that the rolling stock schedule is not changed until the beginning of the disruption, adding the constraints $\widetilde{x}_{t,p} = x_{t,p}$ for each $p \in P$ and trip $t \in T$ ending before the disruption. Finally, the model is extended to express the recovery costs:

$$\lambda \geq c_1 \sum_t \widetilde{x}_{t,\emptyset} + c_2 \sum_{t \in T} \widetilde{E}_t + \sum_{s \in S} \sum_{m \in M} \widetilde{D}_{s,m}, \tag{4.21}$$

$$\widetilde{D}_{s,m} \geq y^\infty_{s,m} - \widetilde{y}^\infty_{s,m}, \qquad \forall s \in S, m \in M, \tag{4.22}$$

$$\widetilde{D}_{s,m} \geq \widetilde{y}^\infty_{s,m} - y^\infty_{s,m}, \qquad \forall s \in S, m \in M, \tag{4.23}$$

$$w_t = \sum \left(z_{t,p,p'} \mid p \to p' \text{ is coupling or uncoupling} \right), \quad \forall t \in T, \tag{4.24}$$

$$\widetilde{w}_t = \sum \left(\widetilde{z}_{t,p,p'} \mid p \to p' \text{ is coupling or uncoupling} \right), \quad \forall t \in T, \tag{4.25}$$

$$\widetilde{E}_t \geq \widetilde{w}_t - w_t, \qquad \forall t \in T, \tag{4.26}$$

$$\widetilde{D}_{s,m} \geq 0, \qquad \forall s \in S, m \in M, \tag{4.27}$$

$$\widetilde{E}_t \geq 0, \qquad \forall t \in T. \tag{4.28}$$

The auxiliary variables $\widetilde{D}$ measure the deviation of the planned end-of-day rolling stock inventories (i.e. that of the nominal solution) from the realised end-of-day rolling stock inventories (i.e. those in the scenario). The value of the auxiliary variable w_t is 0 or 1 depending on whether the nominal solution has a composition change (i.e. coupling or uncoupling of units) after trip t. Similar role is played by $\widetilde{w}_t$ in the scenario. The auxiliary variable $\widetilde{E}_t$ has a value at least 1 if a new shunting is introduced after trip t, i.e. if there was no composition change after trip t in the nominal solution whereas there is one in the recovered solution. The objective penalises the variables $\widetilde{D}$ and $\widetilde{E}$ as well as all variables $\widetilde{x}$ that assign an empty composition to a trip.

4.5 Experimental Results

We implemented the robust scheduling problem (4.8) – (4.11) with the rolling stock (re-)scheduling model described in Section 4.4 for the so called 3000 line of NS. This is an Inter-City line with a closed rolling stock circulation. The instance contains about 400 trips connecting 8 stations, and is served by two rolling stock types with 11 and 24 units, respectively. The 3000 line is one of the medium-sized rolling stock instances of NS.

The nominal problem is based on the actual timetable of NS. The scenarios have been generated artificially using a program of [32], which simulates the decisions of the infrastructure manager about train cancellations, including the successors of the trips after disruption. In that respect, the input data of the scenarios follow the same rules and assumptions as the nominal problem.

As already discussed, the goal of our preliminary computational tests is to investigate whether the suggested optimisation framework can be used at all to assess PoR for our rolling stock scheduling problem. Therefore we restricted ourselves to the solution of LP relaxations.

We implemented two solution methods: (*i*) solving (4.8) – (4.11) directly as a single LP; (*ii*) applying a canonical Benders decomposition Approach. Our computer codes are written in C and run on a personal computer, solving the LPs by ILOG CPLEX 10.0. The master problem has about 14,500 variables, 8,600 constraints and 310,000 non-zeros in the matrix.

The solution approaches have been tested with 2–20 scenarios, implying that the LPs solved by method (*i*) feature 43,000–305,000 variables, 25,000–180,000 constraints and 950,000–6,500,000 non-zeros.

For each number of scenarios, we solved two variants of the problem: Test-I and Test-II. They share the same constraint matrix but differ in the objective function. The cost coefficients are given in Table 4.1. Test-I focuses on service quality (by penalising seat shortages more heavily) while Test-II emphasises efficiency (by penalising carriage kilometres more heavily).

Table 4.1: Coefficients for the nominal objective function as well as for the recovery costs.

Criterion in nominal problem	Test-I	Test-II
seat shortage km	100	50
carriage km	9	100
composition change	5	10
Criterion for recovery	**Test-I**	**Test-II**
cancellation	1,000,000	1,000,000
inventory deviation	20,000	20,000
new shunting	10,000	10,000

The computational results with the two solution approaches are summarised in Table 4.2. It turns out that the huge LPs in method (*i*) are barely solvable. For more than 6 or 7 scenarios, the solution time exceeds our time limit of 1800 seconds. The cases with 10 or more scenarios appear to be far from being solved after several hours of CPU time. The Benders decomposition approach, on the other hand, is able to cope with the problems. After applying 24–640 and 30–360 Benders cuts for Test-I and Test-II, respectively, optimality was reached within the time limit.

The above results prove that, at least for our case study, the general lower bound on PoR

that we propose can be computed within reasonable time.

Table 4.2: The number of applied Benders cuts as well as the running times in seconds for the Benders decomposition approach and for the direct solution of the whole LP (referred to as 'CPLEX') on Test-I and Test-II. A dash indicates the running time of CPLEX exceeding 1800 seconds.

	Test-I			Test-II		
	Benders		CPLEX	Benders		CPLEX
# scens.	# cuts	CPU time	CPU time	# cuts	CPU time	CPU time
2	24	22	122	32	30	75
3	27	27	366	24	25	372
4	140	224	795	100	141	880
5	175	264	1,055	125	175	1,078
6	168	277	1,962	150	209	—
7	210	319	—	140	205	—
8	248	454	—	152	223	—
9	288	603	—	171	278	—
10	320	688	—	190	332	—
11	352	734	—	209	364	—
12	384	798	—	228	400	—
13	325	662	—	234	439	—
14	420	1,014	—	252	499	—
15	450	1,090	—	300	653	—
16	480	1,163	—	320	698	—
17	595	1,710	—	306	642	—
18	540	1,380	—	324	701	—
19	570	1,459	—	342	730	—
20	640	1,840	—	360	816	—

4.6 Summary and Future Research

In this chapter we summarised our understanding of the Price of Recoverability. In addition, we proposed a mathematical programming approach to compute a lower bound on the Price of Recoverability for real-life railway scheduling problems.

A Benders decomposition approach has been implemented for a medium-sized rolling stock scheduling problem of NS. The preliminary computational results indicate the problem is widely tractable with up to 20 disruption scenarios. Note that the whole problem (a single LP) cannot be solved within several hours even with just 10 scenarios.

In order to improve the proposed method, we will go on with more thorough computational tests. First, the preliminary computations concern the LP relaxation of the rolling stock scheduling problem; we are going to study the original MILP models as well. Second, the current way of selecting the Benders cuts is very simple; we are going to evaluate the effect of more sophisticated cut selection methods. Third, Benders decomposition is not the only possible approach for solving (4.8) – (4.11). In our future research we are going to explore other mathematical programming techniques, such as convex optimisation (e.g. through the

subgradient algorithm). Last but not least, we are going to investigate implications of the Price of Recoverability to railway practice.

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).

4.7 Appendix 1

In this section we will show, with more details, the Benders decomposition method used.

Let the following linear program be the master problem and one subproblem together. Here I denotes the identity matrix, and M_i are matrices of appropriate size.

$$
\begin{array}{cccccccccccccc}
\lambda & x & z & I^0 & I^\infty & \widetilde{x} & \widetilde{z} & \widetilde{I}^\infty & w & \widetilde{w} & \widetilde{E}\geq 0 & \widetilde{D}\geq 0 & & \\
 & M_1 & M_2 & M_3 & M_4 & & & & & & & & \geq b & (4.29)\\
M_5 & M_6 & M_7 & M_8 & M_9 & & & & & & & & \geq v & (4.30)\\
 & & & & & M_{10} & M_{11} & M_{12} & & & & & \geq \widetilde{b} & (4.31)\\
 & & & & I & & & -I & & & & I & \geq 0 & (4.32)\\
 & & & & -I & & & I & & & & I & \geq 0 & (4.33)\\
 & & M_{13} & & & & & & I & & & & = 0 & (4.34)\\
 & & & & & & M_{14} & & & I & & & = 0 & (4.35)\\
 & & & & & & & & I & -I & I & & \geq 0 & (4.36)\\
1 & & & & & M_{15} & & & & & M_{16} & M_{17} & \geq 0 & (4.37)
\end{array}
$$

Notice that M_{15}, M_{16} and M_{17} are actually row vectors:

- $M_{15} \in \mathbb{R}^{\text{Trips}\times\text{Compos}}$, the entries are $-c_{\text{cancel}}$ if the composition is empty, 0 otherwise; here c_{cancel} is the (positive and huge) penalty for cancelling a trip;
- $M_{16} \in \mathbb{R}^{\text{Trip}}$, each entry is $-c_{\text{new}}$, the negative of the penalty for new shunting;
- $M_{17} \in \mathbb{R}^{\text{Station}\times\text{Type}}$, each entry is $-c_{\text{dev}}$, the negative of the penalty for deviation.

(4.29) is the master problem as formulated till now. (4.30) are all the Benders cuts that have been found so far (at the beginning we have $\lambda \geq 0$). (4.31) is the basic part of the subproblem which is similar to (4.29). (4.32) and (4.33) set the final inventory deviation variables $\widetilde{D}$. (4.34) and (4.35) set the flags w and $\widetilde{w}$ whether or not shunting (uncoupling or coupling) takes place after trip t. (4.36) is for deciding whether new shunting is introduced after trip t. Finally, (4.37) states that the recovery costs are no higher than λ.

Suppose we have a master solution, and we want to decide if there is a recovery plan with costs no higher than λ, actually: if there is a *fractional* recovery plan. That is, we ask if the following LP with fixed $(\lambda, x, z, I^0, I^\infty)$ is feasible:

$$
\begin{array}{cccccccccccccc}
\text{(dual:)} & \lambda & x & z & I^0 & I^\infty & \widetilde{x} & \widetilde{z} & \widetilde{I}^\infty & w & \widetilde{w} & \widetilde{E}\geq 0 & \widetilde{D}\geq 0 & & \\
(\alpha\geq 0) & & & & & & M_{10} & M_{11} & M_{12} & & & & & \geq \widetilde{b} & (4.38)\\
(\beta\geq 0) & & & & & I & & & -I & & & & I & \geq 0 & (4.39)\\
(\gamma\geq 0) & & & & & -I & & & I & & & & I & \geq 0 & (4.40)\\
(\delta) & & & M_{13} & & & & & & I & & & & = 0 & (4.41)\\
(\varepsilon) & & & & & & & M_{14} & & & I & & & = 0 & (4.42)\\
(\zeta\geq 0) & & & & & & & & & I & -I & I & & \geq 0 & (4.43)\\
(\eta\geq 0) & 1 & & & & & M_{15} & & & & & M_{16} & M_{17} & \geq 0 & (4.44)
\end{array}
$$

This is just a copy of the inequalities above; we indicate at the left hand side the dual variables.

The infeasibility of (4.38) – (4.44) means that the recovery of the master solution costs more than λ. Then we have to cut off this master solution space.

If we take any linear combinations of (4.38) – (4.44) with weights $\alpha, \ldots, \eta$ (respecting non-negativities), then the resulting inequality is a valid cut for the large optimisation problem, i.e. for (4.29) – (4.37). If we are lucky, then all tilde-variables in such a cut have 0 coefficients, and the resulting cut is expressed by the master variables only. If we are even more lucky, then such a master-cut is violated by the wrong master solution.

4.7.1 Subproblem for a fixed master solution

Suppose we have a(n optimal) master solution $(\lambda^\star, x^\star, z^\star, I^{\infty\star})$ and that (4.38) – (4.44) is infeasible. We reformulate the last linear program to have $\leq$ signs, and by multiplying (4.41) and (4.42) by -1:

$$
\begin{array}{rccccccccll}
\text{(dual:)} & \widetilde{x} & & \widetilde{z} & \widetilde{I}^\infty & w & \widetilde{w} & \widetilde{E} \geq 0 & \widetilde{D} \geq 0 & & \\
(\alpha \geq 0) & -M_{10} & & -M_{11} & -M_{12} & & & & & \leq -\widetilde{b} & (4.45) \\
(\beta \geq 0) & & & & I & & & & -I & \leq I^{\infty\star} & (4.46) \\
(\gamma \geq 0) & & & & -I & & & & -I & \leq -I^{\infty\star} & (4.47) \\
(\delta) & & & & & -I & & & & = M_{13} z^\star & (4.48) \\
(\varepsilon) & & & -M_{14} & & & -I & & & = 0 & (4.49) \\
(\zeta \geq 0) & & & & & -I & I & -I & & \leq 0 & (4.50) \\
(\eta \geq 0) & -M_{15} & & & & & & -M_{16} & -M_{17} & \leq \lambda^\star & (4.51)
\end{array}
$$

By Farkas, if this is infeasible, then there exist $\alpha, \ldots, \eta$ with $\alpha, \beta, \gamma, \zeta, \eta \geq 0$ and with

$$
\begin{array}{cccccccccll}
-\alpha^\mathsf{T} M_{10} & & & & & & & & -\eta^\mathsf{T} M_{15} & = 0 & (4.52) \\
-\alpha^\mathsf{T} M_{11} & & & & & & -\varepsilon^\mathsf{T} M_{14} & & & = 0 & (4.53) \\
-\alpha^\mathsf{T} M_{12} & & +\beta^\mathsf{T} & & -\gamma^\mathsf{T} & & & & & = 0 & (4.54) \\
& & & & & -\delta^\mathsf{T} & & -\zeta^\mathsf{T} & & = 0 & (4.55) \\
& & & & & & -\varepsilon^\mathsf{T} & +\zeta^\mathsf{T} & & = 0 & (4.56) \\
& & & & & & & -\zeta^\mathsf{T} & -\eta^\mathsf{T} M_{16} & \geq 0 & (4.57) \\
& & -\beta^\mathsf{T} & & -\gamma^\mathsf{T} & & & & -\eta^\mathsf{T} M_{17} & \geq 0 & (4.58) \\
-\alpha^\mathsf{T}\widetilde{b} & & +\beta^\mathsf{T} I^{\infty\star} & & -\gamma^\mathsf{T} I^{\infty\star} & +\delta^\mathsf{T} M_{13} z^\star & & & +\eta^\mathsf{T}\lambda^\star & < 0 & (4.59)
\end{array}
$$

4.7.2 Getting the Benders cut

Considering the vectors $(\alpha, \ldots, \eta)$ obtained from the Farkas dual, we multiply (4.38) – (4.44) by them, and we obtain:

$$\eta^\mathsf{T}\lambda + \delta^\mathsf{T} M_{13} z + (\beta^\mathsf{T} - \gamma^\mathsf{T}) I^\infty + (\zeta^\mathsf{T} + \eta^\mathsf{T} M_{16})\widetilde{E} + (\beta^\mathsf{T} + \gamma^\mathsf{T} + \eta^\mathsf{T} M_{17})\widetilde{D} \geq \alpha^\mathsf{T}\widetilde{b}. \quad (4.60)$$

Observe that $\zeta^\mathsf{T}+\eta^\mathsf{T}M_{16} \leq 0$ (by (4.57)) and $\widetilde{E} \geq 0$. Also observe that $\beta^\mathsf{T}+\gamma^\mathsf{T}+\eta^\mathsf{T}M_{17} \leq 0$ (see (4.58)) and $\widetilde{D} \geq 0$. Therefore (4.60) can be weakened to

$$\eta^\mathsf{T}\lambda + \delta^\mathsf{T}M_{13}z + (\beta^\mathsf{T} - \gamma^\mathsf{T})I^\infty \geq \alpha^\mathsf{T}\widetilde{b}. \tag{4.61}$$

Finally, (4.59) shows that our master solution $(\lambda^\star, x^\star, z^\star, I^{0\star}, I^{\infty\star})$ violates the cut (4.61).

4.7.3 Solving the subproblem as an LP

The feasibility check of (4.45) – (4.51) could be slightly tricky, partly due to (4.51) which is a nasty constraint. Therefore we may wish to convert it to an LP: solve (4.45) – (4.50), and minimise the left-hand side of (4.51). Just for the sake of convenience, let us turn it to a maximisation problem:

(dual:)	$\widetilde{x}$	$\widetilde{z}$	$\widetilde{I}^\infty$	w	$\widetilde{w}$	$\widetilde{E} \geq 0$	$\widetilde{D} \geq 0$		
$(\alpha \geq 0)$	$-M_{10}$	$-M_{11}$	$-M_{12}$					$\leq -\widetilde{b}$	(4.62)
$(\beta \geq 0)$			I				$-I$	$\leq I^{\infty\star}$	(4.63)
$(\gamma \geq 0)$			$-I$				$-I$	$\leq -I^{\infty\star}$	(4.64)
(δ)				$-I$				$= M_{13}z^\star$	(4.65)
(ε)		$-M_{14}$			$-I$			$= 0$	(4.66)
$(\zeta \geq 0)$				$-I$	I	$-I$		≤ 0	(4.67)
	M_{15}					M_{16}	M_{17}	$\to \max$	(4.68)

The question is whether or not the optimal objective value is at least $-\lambda^\star$. The dual LP looks as follows (the dual constraints are to be read column-wise).

	$(\widetilde{x})$	$(\widetilde{z})$	$(\widetilde{I}^\infty)$	(w)	$(\widetilde{w})$	$(\widetilde{E} \geq 0)$	$(\widetilde{D} \geq 0)$		
$\alpha \geq 0$	$-M_{10}$	$-M_{11}$	$-M_{12}$					$-\widetilde{b}$	(4.69)
$\beta \geq 0$			I				$-I$	$I^{\infty\star}$	(4.70)
$\gamma \geq 0$			$-I$				$-I$	$-I^{\infty\star}$	(4.71)
δ				$-I$				$M_{13}z^\star$	(4.72)
ε		$-M_{14}$			$-I$				(4.73)
$\zeta \geq 0$				$-I$	I	$-I$			(4.74)
	$\leq$					$\geq$	$\geq$	$\downarrow$	
	M_{15}	0	0	0	0	M_{16}	M_{17}	$\min$	

Consider any dual optimal solution $\alpha, \ldots, \zeta$, and choose $\eta = 1$. Then again, take the linear combination of the inequalities (4.38) – (4.44) with these weights. Just like in the previous section, we have the Benders cut

$$\lambda + \delta^\mathsf{T}M_{13}z + (\beta^\mathsf{T} - \gamma^\mathsf{T})I^\infty \geq \alpha^\mathsf{T}\widetilde{b}. \tag{4.75}$$

Also, $(\lambda^\star, x^\star, z^\star, I^{\infty\star})$ violates this cut. Indeed, (4.45) – (4.51) is infeasible if and only if if the objective value is smaller than $-\lambda^\star$, i.e. iff

$$-\alpha^\mathsf{T}\widetilde{b} + \beta^\mathsf{T}I^{\infty\star} - \gamma^\mathsf{T}I^{\infty\star} + \delta^\mathsf{T}M_{13}z^\star < -\lambda^\star. \tag{4.76}$$

4.7.4 Miscellaneous observations

The Benders cut (4.75) looks the same as the Benders cut (4.61), except that the value of η is set to 1. Whenever the Farkas dual provided a cut with $\eta > 0$, we can scale it to 1. But what if the Farkas dual gives $\eta = 0$?

Looking at (4.52) – (4.59) and supposing $\eta = 0$, then using the non-negativity of the variables, we have $\beta = \gamma = \delta = \varepsilon = \zeta = 0$. The remaining program is

$$-\alpha^{\mathsf{T}} M_{10} = 0 \tag{4.77}$$
$$-\alpha^{\mathsf{T}} M_{11} = 0 \tag{4.78}$$
$$-\alpha^{\mathsf{T}} M_{12} = 0 \tag{4.79}$$
$$\alpha \geq 0 \tag{4.80}$$
$$-\alpha^{\mathsf{T}} \widetilde{b} < 0. \tag{4.81}$$

This is nothing but the Farkas dual of

$$-M_{10}\widetilde{x} - M_{11}\widetilde{z} - M_{12}\widetilde{I}^{\infty} \leq -\widetilde{b} \tag{4.82}$$

which is the same as (4.38): the core rolling stock model for the subproblem. This can never be infeasible (we can cancel everything, after all), so we cannot ever get $\eta = 0$.

The same considerations are relevant for the primal-dual LP's. Does there always exist an optimal dual solution? Yes, there does. The primal LP (4.62) – (4.68) always has a feasible solution (namely cancelling everything). Furthermore, the objective of the maximisation is the negative of the recovery costs, and it cannot be positive. Since the objective function is bounded from above, there always exists a dual solution. Finally, by primal feasibility, the dual objective is bounded from below.

4.7.5 Possible improvements

In the previous section we have seen that if the subproblem (4.38) – (4.44) is infeasible, then each optimal solution of the dual LP gives rise to a Benders cut. This may lead some improvements of the Benders process by selecting *several* Benders cuts for each single infeasible subproblem. The reason is that one might need to re-optimise the master problem less frequently.

As for the implementation of the hoped improvement:

1. One should solve the dual LP, let the optimal solution be $(\alpha', \ldots, \eta')$ and the optimal value be ϕ (recall that $\phi < \lambda'$).

2. One should add the constraint

$$-\alpha^{\mathsf{T}} \widetilde{b} + \beta^{\mathsf{T}} I^{\infty\star} - \gamma^{\mathsf{T}} I^{\infty\star} + \delta^{\mathsf{T}} M_{13} z^{\star} = \phi \tag{4.83}$$

   to enforce solutions that are optimal for (4.69) – (4.74). Let us call this the extended dual LP.

3. The extended dual LP should be solved for any objective to get various solutions $(\alpha, \ldots, \eta)$. It could be essential to use $(\alpha', \ldots, \eta')$ as a starting solution.

The solution space of the extended dual can actually be enlarged by replacing (4.83) e.g. by the following:

$$-\alpha^{\mathsf{T}}\widetilde{b} + \beta^{\mathsf{T}} I^{\infty\star} - \gamma^{\mathsf{T}} I^{\infty\star} + \delta^{\mathsf{T}} M_{13} z^{\star} \leq \frac{\phi + \lambda^{\star}}{2} \qquad \text{(which is } < \lambda^{\star}\text{)}. \tag{4.84}$$

Indeed, we can generate the Benders cut just like in (4.75), and it will be violated by $(\lambda^{\star}, x^{\star}, z^{\star}, I^{0\star}, I^{\infty\star})$ just like above. This may help to find Benders cuts very different from the ones corresponding to the optimal solutions of the "basic" dual LP (4.69) – (4.74).

Chapter 5

Recovery-Robust Platforming by Network Buffering

We aim at assigning platforms, arrival and departure routes to trains at a station in such a way that the total delay in the system caused by some inevitable, small, seminal disturbances in every-day operations is guaranteed to stay below some minimised bound. To this end we apply the concept of *recoverable robustness*, in particular *network buffering*, to the standard *train platforming problem*. The case study of two major railway stations with high load shows that recovery-robust platforming plans achieve maximal throughput of trains, while substantially reducing, in some cases even halving, the propagated delay in the system.

5.1 Introduction

Assigning and routing trains to platforms in a station is an independent planning step for railway operations optimization. Often, as in the real world data of this study, the capacities of the stations are bottlenecks of the system. Through such bottlenecks even the small, daily, and unavoidable delays of some trains can spread heavily onto the whole system. Therefore, the goal is a platforming plan yielding a high throughput of trains in the station while limiting the spread-on delays.

Robust optimization is an approach to optimization under uncertainty with mainly two advantages. First, robust models often stay tractable both in theory and practice even for large scale instances. Second, robust solutions have guaranteed quality for a large set of likely scenarios. They are not only good on average, but reliable in all likely situations. On the down side, robust solutions tend to be over-conservative in many applications, including the platforming problem.

Recoverable Robustness [28] has been developed to overcome this intrinsic weakness of classical robustness. A classical robust solution is feasible in every likely scenario. In contrast, a recovery-robust solution can be recovered in every likely scenario with an a priori limited effort. Classical robust solutions for platforming need to have excessively large time buffers between *each* pair of trains using a common resource (a platform or a path). In a recovery-robust platforming the buffers are cautiously distributed in the system to ensure that the total delay stays below a certain threshold in every likely scenario.

Recoverable robustness is very well suited for platforming. In particular, robust platforming can be formulated as a robust network buffering problem (cf. [28]), which is a special case

of recoverable robustness. This allows for a tractable model, which can be tackled by existing solving techniques for the original optimization problem.

To the best of our knowledge, moving from theory to practice, this is the first real-world case study on recoverable robustness by network buffering, and shows its effectiveness.

The resulting recovery robust platforming produces, in some tight cases, even 51% less delay than a standard platforming with the same nominal quality. Thus, using this method one can substantially decrease the spread-on effect of delays while maintaining the throughput of the station.

5.2 Recovery-Robust Platforming

Planning and optimization of railway operations proceeds in several subsequent steps. A classical step following immediately the timetabling step is called *platforming*, and considers a single train station. The trains have roughly fixed arrival and departure times at the station, calculated in the timetabling phase. The task of platforming is to assign the trains to the tracks on the station and choose paths to arrive at and depart from these tracks.

In many cases, as in our real-world study on two main stations of the Italian railway network, the platforming capacity of a station is a bottleneck of the whole system, so tight planning is required. As a consequence, even small disturbances to the schedule can widely affect the smooth operation of the system. Therefore, in this study we construct a platforming plan which can be operated with only a limited amount of total delay and without re-planning of tracks or paths in all likely scenarios, i.e., in those scenarios with a limited amount of small disturbances. In other words, we seek a robust platforming.

The terms 'robust' and 'robust optimization' are not used consistently in the literature. The notions of robustness and recoverable robustness used here are in the tradition of robust optimization originating from Soyster [33], and best surveyed by a recent special issue [1] of a leading journal in optimization. This field of research is a special branch of stochastic programming.

Robust optimization finds best solutions, which are feasible for all likely scenarios. This has the advantage that no knowledge of the underlying distribution is required, taking into account that acquiring data about the distribution is not always possible in practice. Moreover, robust models are usually easier to solve, because the solver need not handle distributions, nor look explicitly at each scenario to assess the objective. This is of particular relevance for the optimization of large-scale systems. Finally, robust solutions are guaranteed to be feasible in all likely scenarios. In many applications this is more desirable than , e.g., a satisfying performance of the solution on average. Hence, in our case robust platforming will provide for smooth operations in all cases except those that feature extraordinarily high disturbances.

On the down side of the classical robust approach, which we call *strict robustness*, is its excessive conservatism in some applications. This is due to two reasons. First, strict robust solutions must cope with every likely scenario without any recovery. In many applications, limited and simple means of recovery are provided quite naturally, and excluding them from the model destroys the vital flexibility of the system. For example, a timetable without the flexibility of small delays for some events must be planned very loosely in advance even to cope with small disturbances. Second, strict robustness is unable to account for limits to the sum of all disturbances, whereas, in practice, it can often be inferred that the number of constraints affected by disturbances is limited. For example, only a few train rides experience seminal

disturbances. From a strict robust perspective, hedging against those limited scenarios is equivalent to hedging against scenarios in which all train rides have seminal disturbances.

The notion of *recoverable robustness* has been developed inter alia to overcome these weaknesses of strict robustness. Informally speaking, a solution to an optimization problem is called recovery robust if it is likely that it can be adjusted to the actual scenario by limited changes. Thus, a recovery-robust solution also provides a service guarantee. For instance, for all scenarios with moderate disturbances, a recovery-robust platforming solution is guaranteed to be adjustable by simple means and at an a priori limited cost, i.e., the total delay will not exceed a certain a priori fixed budget. Moreover, recoverable robustness can fulfill its guarantee at a lower cost in the nominal objective, because it can take into account restricted scenario sets for uncertainty on the right-hand side. For some types of recovery-robust problems generic, tractable models exist. In particular, recovery-robust platforming is a tractable case as an example of robust network buffering.

Platforming instances are large in size and create huge scenario spaces due to the many activities that can be disturbed independently. The joint distribution of the disturbances depends on the timetable, which is usually released together with the platforming. Therefore, historic, exhaustive data of the joint distribution is often not available. The large scale of the instances and the data situation for the distributions make a *robust* model the favorable method to cope with the uncertainty.

On the other hand, platforming problems feature both causes for over-conservatism in classical robust models. Platforming naturally allows for a limited, simple recovery, i.e., a limited amount of delays. And it makes sense to assume that only a limited number of trains will reach the station initially delayed, or experience a seminal disturbance on the platform before departure. We will define the set of likely scenarios in this spirit.

An advantageous feature of the robust platforming problem is that disturbances only occur on the right hand side of the problem. This enables the use of the robust network buffering, which yields tractable recovery-robust models. Roughly speaking 'tractable' means that, out of the infinite (uncountable) set of scenarios, we can restrict attention to a finite set in modeling the problem. Formally, this rests on the theorem of [28], illustrated in Chapter 3.

Robust Network Buffering. We are interested in the special case in which the recovery problem is a (delay) propagation in some directed graph N with n vertices and m arcs, which is buffered on the arcs by means of f against disturbances on the arcs b. By denoting by $A(N)$ the set of arcs in N, we get:

$$\begin{aligned} &\min_{f \in P} c(f) \\ \text{s.t. } \forall a \in A(N)\ \exists y^a \in \mathbb{R}^{|A(N)|}: & \\ & f_{(i,j)} + y^a_i - y^a_i \geq \Delta \cdot \chi_a((i,j)), \qquad (i,j) \in A(N) \\ & D - d' y^a \geq 0 \end{aligned}$$

where χ_a is the characteristic function of arc a, i.e., $\chi_a((i,j))$ is equal to 1 when $a = (i,j)$ and zero otherwise.

Note that the arc set A serves both as a representation of the scenario set and as the set of edges along which the delay y on the nodes propagates until it is absorbed by the buffers

on the arcs f. A recovery variable y with arc a in the superscript and node i in the subscript expresses the delay of node i in the scenario where edge a is maximally disturbed, notably by the value Δ.

This setting is called in [28] the (general, fractional) robust network buffering problem. We will phrase the robust platforming problem as a robust network buffering problem in Section 5.4 by formalizing the link between the platforming and the network buffers. Then we can apply the theorem of [28] to get a compact formulation as an integer linear program. In this process, the nominal part of the robust program, namely, $\min_{f \in P} c(f)$, will be played by the deterministic platforming problem. The formulation of the latter can be supplemented with the robust network buffering approach without changing the platforming model. In other words, the robustness is added, so to say, from the outside. This allows to use the same, specialized solving procedures (cf. [11]) as in the deterministic platforming problem.

5.3 The Train Platforming Problem

We now describe in detail the deterministic Train Platforming Problem (TPP) (cf. [11]). Time is discretized in minutes and we split the whole day, i.e., 1440 minutes, into seven time windows. The decision variables correspond to so-called *patterns*. For each train t, we are given a set of patterns $\mathcal{P}_t$, each one encoding a stopping platform, an arrival and a departure path connecting respectively the arrival and departure direction of train t to the given platform. The input of the TPP also contains an incompatibility graph with the set of all patterns $\bigcup_{t \in T} \mathcal{P}_t$ as node set. Each train t has an *ideal arrival time* v_t^{a} and an *ideal departure time* v_t^{d} at respectively from the platform. We say the pattern occupies platform b for the interval $[v_t^{\mathrm{a}} - h, v_t^{\mathrm{d}} + h]$, where h is a buffer time called *headway* introduced for safety reasons. Moreover, the pattern occupies the arrival or departure paths from the time it enters until the time it leaves the path. Hence there is no headway here. Two patterns $P_1 \in \mathcal{P}_{t_1}$ and $P_2 \in \mathcal{P}_{t_2}$ are incompatible if either their platform occupation intervals overlap for a time window of duration > 0, or if they occupy so-called incompatible paths (which are arrival/departure paths with nonempty physical intersection) for a time window of duration $> \pi$, where π is a so-called *threshold*.

The structure of the railway station may not allow the complete platforming of all trains in timetable. Therefore in the model we allow the use of *dummy* platforms, which can be reached and left without path conflicts and are meant to measure the lack of capacity in the train station.

Each pattern has a cost $c_{t,P}$, which is either zero, or equal to a penalty for using a non-preference platform, or equal to a very high penalty in case of a dummy platform. This penalty is so high that a dummy platform is only used if no physically feasible platforming is possible. This is the deterministic part of our objective function. The objective used in [11] has one additional term measuring the number of minutes of incompatibility over the paths (while each conflict must still be below π). This can be seen as a heuristic approach towards reliable planning. We omit this term since we supplement the model with an exact robust approach.

The resulting ILP formulation of the problem uses a binary variable $x_{t,P}$ for each $t \in T$ and $P \in \mathcal{P}_t$, indicating whether train t is assigned pattern P:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} \tag{5.1}$$

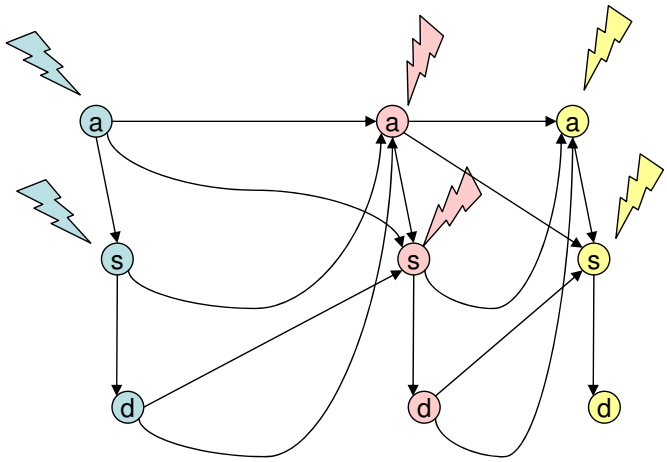

Figure 5.1: Delay Propagation Network.

s.t.

$$\sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T \tag{5.2}$$

$$\sum_{(t_1,P_1) \in K} x_{t_1,P_1} + \sum_{(t_2,P_2) \in K} x_{t_2,P_2} \leq 1, \qquad (t_1,t_2) \in T^2, K \in \mathcal{K}(t_1,t_2) \tag{5.3}$$

$$x_{t,P} \in \{0,1\}, \qquad b \in B,\ t \in T,\ P \in \mathcal{P}_t \tag{5.4}$$

where T^2 denotes the set of train pairs and $\mathcal{K}(t_1,t_2)$ the collection of cliques containing only incompatible patterns in $\mathcal{P}_{t_1} \cup \mathcal{P}_{t_2}$. Constraints (5.2) guarantee that each train is assigned a pattern and constraints (5.3) forbid the assignment of patterns that are pairwise incompatible.

Note that requiring all clique inequalities, i.e.,

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

with $\mathcal{K}$ set of all cliques in the pattern-incompatibility graph, gives a stronger LP relaxation, but those constraints are NP-complete (and practically hard) to separate.

For the applications we are aware of, including our case study, the overall number of patterns $\sum_{t \in T} |\mathcal{P}_t|$ allows us to handle explicitly all of them.

5.4 Platforming and Buffering

The recovery-robust TPP model combines the deterministic model described in Section 5.3 with the network buffering approach of Section 5.2. Therefore, the full recovery-robust TPP model consists of three blocks, the deterministic TPP model, the network buffering model, and the linking constraints.

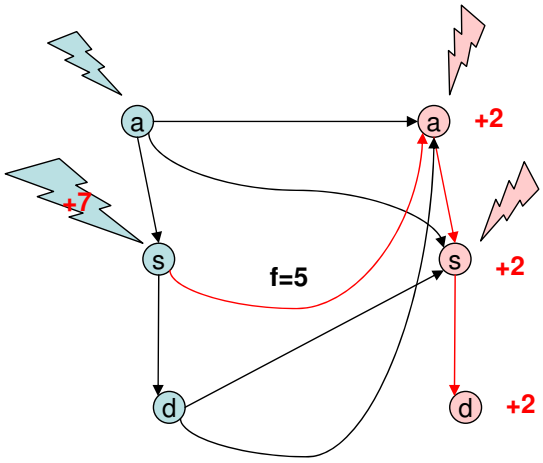

Figure 5.2: Delay Propagation Network Example 2.

Delay propagation network. The source of delay considered in our model is the presence of inevitable, small, seminal disturbances in daily operations. In particular, trains may reach the station area later than scheduled, or stops may take longer than planned. The platforming gives rise in a natural way to a network in which the delay caused by those disturbances propagates. This *delay propagation network* is a directed acyclic graph in which each vertex represents the delay of a particular train for a particular resource, see Figure 5.1. Each train has three associated vertices in this graph: (i) a for the arrival path, (ii) s for the stopping platform, and (iii) d for the departure path, corresponding to the delay (with respect to the nominal schedule) with which it will free up each of the three resources assigned to it. Naturally, two vertices are connected by an arc whenever the corresponding train-resource pairs may cause delay propagation. This happens, for example, if two nodes correspond to the same platform occupied by different trains: a delay in freeing up the platform for the first train may propagate to a delay in freeing up the same platform for the second.

Example 2. *Assume in the nominal schedule the first train frees up the platform at 10:00, the second train occupies it at 10:05 and frees it up at 10:10. Then a delay of more than 5 minutes for the first train results in a delay also for the second. See Figure 5.2.*

Something similar applies to incompatible paths. In detail we get the following five types of arcs in the delay propagation network N:

1. Whenever the arrival path of train t is incompatible with the arrival path of train t', an arc of type (a_t, a'_t) is present in $A(N)$.

2. Whenever the arrival path of train t is incompatible with the departure path of train t', an arc of type (a_t, s'_t) is present in $A(N)$. This is because if the departure path of train t' is blocked by the arrival path of train t, train t' will not free-up its stopping platform. So the delay propagates to the release time of the stopping platform of train t'.

3. Whenever trains t and t' share the same platform, an arc of type (s_t, a'_t) is present in

$A(N)$. In this case the stopping platform for train t' is not available, so t' will be forced to occupy its arrival path beyond the nominal release time.

4. Whenever the departure path of train t is incompatible with the arrival path of train t', an arc of type (d_t, a'_t) is present in $A(N)$.

5. Whenever the departure path of train t is incompatible with the departure path of train t', an arc of type (d_t, s'_t) is present in $A(N)$. This is because if the departure path of train t' is blocked by the departure path of train t, train t' will not free-up its stopping platform. So the delay propagates to the release time of the stopping platform of train t'.

Mind that the orientation of an arc depends on the nominal release times of the resources. In other words, given a timetable we construct arc (i, j) in $A(N)$ if the nominal release time of resource i is no later than the nominal release time of resource j.

Linking constraints. The third block establishes the link between the choice of the patterns and the buffers on the arcs of N. In fact, 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 Example 2. Clearly the propagation, i.e., the result of the recovery sub-model, depends on the solution of the planning sub-model. In other words, once the patterns are assigned to the corresponding trains, the buffer values on the arcs $A(N)$ of the delay propagation network are defined.

Unfortunately, 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 fact the straightforward formulation to link the buffer value of a given arc $a \in A(N)$ associated with train pair $(t_1, t_2) \in T^2$ to the choice of patterns is the following:

$$\sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{P_1,P_2,a}\ x_{t_1,P_1}\ x_{t_2,P_2}$$

where c_{a,P_1,P_2} is a constant associated to arc a and to the corresponding choice of patterns (P_1, P_2) for trains (t_1, t_2). This situation is similar to the one encountered in dealing with the penalties in the original nominal problem in [11]. Rather than using a classical linearisation, with many variables and weak constraints, as in [11] we proceed as follows.

We introduce the $|A(N)|$ additional continuous variables f_a for $a \in A(N)$ associated with two resources of the pair of trains $(t_1, t_2) \in T^2$ each representing the term

$$\sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{P_1,P_2,a}\ x_{t_1,P_1}\ x_{t_2,P_2}.$$

Taking into account the assignment constraints (5.2) and observing that there are up to $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ possible values for f_a, we consider the simple polyhedron in $\mathbb{R}^{|\mathcal{P}_{t_1}|+|\mathcal{P}_{t_2}|+1}$ corresponding to the convex hull of the $|\mathcal{P}_1||\mathcal{P}_{t_2}|$ possible values taken at the same time by vectors $(x_{t_1,P_1})_{P_1 \in \mathcal{P}_{t_1}}$, $(x_{t_2,P_2})_{P_2 \in \mathcal{P}_{t_2}}$ and by variable f_a in a solution:

$$Q_a := \mathrm{conv}\{(e_{P_1}, e_{P_2}, c_{P_1,P_2,a}) : P_1 \in \mathcal{P}_{t_1},\ P_2 \in \mathcal{P}_{t_2}\}, \tag{5.5}$$

where, with a slight abuse of notation, for $i = 1, 2$, we let e_{P_i} denote the binary vector in $\mathbb{R}^{|\mathcal{P}_i|}$ with the P_i-th component equal to 1 and all other components equal to 0.

Among the valid inequalities for Q_a, we are interested in those of the form

$$f_a \leq \sum_{P_1 \in \mathcal{P}_{t_1}} \alpha^a_{P_1}\ x_{t_1,P_1} + \sum_{P_2 \in \mathcal{P}_{t_2}} \beta^a_{P_2}\ x_{t_2,P_2} - \gamma^a. \tag{5.6}$$

We let $\mathcal{F}_a \subseteq \mathbb{R}^{|\mathcal{P}_1|+|\mathcal{P}_{t_2}|+1}$ be the collection of vectors (α, β, γ) such that inequality (5.6) is valid for Q_a and not dominated by other valid inequalities.

Robust ILP model. The complete model looks as follows:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + D \tag{5.7}$$

s.t.

$$\sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T \tag{5.8}$$

$$\sum_{(t_1,P_1) \in K} x_{t_1,P_1} + \sum_{(t_2,P_2) \in K} x_{t_2,P_2} \leq 1, \qquad (t_1, t_2) \in T^2, K \in \mathcal{K}(t_1, t_2) \tag{5.9}$$

$$D \geq \sum_{t \in T} (a^\xi_t + s^\xi_t + d^\xi_t), \qquad \xi \in \{\delta_t | t \in T\} \cup \{\delta'_t | t \in T\} \tag{5.10}$$

$$a^\xi_t \geq \delta^\xi_t, \qquad t \in T \tag{5.11}$$

$$s^\xi_t \geq a^\xi_t + \delta'^\xi_t, \qquad t \in T \tag{5.12}$$

$$d^\xi_t \geq s^\xi_t, \qquad t \in T \tag{5.13}$$

$$m^\xi_{t_2} \geq h^\xi_{t_1} - f(h_{t_1}, m_{t_2}), \qquad a = (h_{t_1}, m_{t_2}) \in A(N) \tag{5.14}$$

$$f_a \leq \sum_{P_1 \in \mathcal{P}_{\sqcup_\infty}} \alpha^a_{P_1} x_{t_1,P_1} + \sum_{P_2 \in \mathcal{P}_{\sqcup_\in}} \beta^a_{P_2} x_{t_2,P_2} - \gamma^a, \qquad a \in A(N),\ (\alpha, \beta, \gamma) \in \mathcal{F}_a \tag{5.15}$$

Variables:

$$x_P \in \{0, 1\}$$
$$f, a^\xi_t, d^\xi_t \geq 0$$

Constants:

$$\delta'^\xi_t, \delta^\xi_t \in \{0, \Delta\}$$

$$\delta_t^{\xi} = \Delta \Leftrightarrow \xi = \delta_t$$

$$\delta_t^{'\xi} = \Delta \Leftrightarrow \xi = \delta'_t$$

Notice that the objective function of the robust platforming model consists of the deterministic component plus the budget variable D, which models the maximal amount of total delay propagated through the network over all scenarios. Δ represents the maximum total disturbance we consider likely, hence this expresses a limit on our scenario space. Constants δ_t^{ξ} and $\delta_t^{'\xi}$ are respectively the seminal disturbances on the arrival time and stopping activity in scenario ξ. By theorem of [28] it suffices to consider the scenarios with a single disturbance of size Δ.

5.5 Solving the Program

Our overall method is the branch-and-bound algorithm in [11], based on variable pricing and on the separation of Constraints (5.9) and (5.15). Moreover, branching is aimed at quickly finding a good feasible solution.

Notice that the Constraints (5.15) that we separate are valid *locally*, i.e., for the subset of variables that are currently in the model. This means that when a new *priced-out* variable is added, we need to update all the buffer constraints according to the coefficients calculated in the pricing phase. Whereas the Constraints (5.9) are valid *globally*. So it is not necessary to update them during the pricing. Clearly, if the update takes place, by maintaining the cliques *maximal* the model gets stronger.

Following [11], the separation of Constraints (5.15) is done by a sort of polyhedral brute force, given that, for each pair of trains t_1, t_2, and for each arc $a \in A(N)$ the number of vertices in $Q_{t_1,t_2,a}$ is small. Specifically, $Q_{t_1,t_2,a}$ has $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ vertices and lies in $\mathbb{R}^{|\mathcal{P}_{t_1}|+|\mathcal{P}_{t_2}|+1}$, so we can separate over it by solving an LP with $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ variables and $|\mathcal{P}_{t_1}|+|\mathcal{P}_{t_2}|+1$ constraints.

5.6 Experimental Results

Our method was implemented in ANSI C and tested on a PC Pentium 4, 3.2 GHz, with a 2 GB RAM. The instances of our case study come from Rete Ferroviaria Italiana, the Italian Infrastructure Manager. This is the same benchmark data used in the deterministic study in [11]. In this chapter, we confine ourselves to the stations of Palermo Centrale and Genova Piazza Principe.

In Tables (5.1-5.3) and (5.4-5.6) we report respectively the results for seven time windows of Palermo Centrale and Genova Piazza Principe and different thresholds π. These results (column "Diff. D") are plotted in Figures (5.3-5.8). By 'nom' we refer to solutions optimized for the deterministic TPP objective (cf. [11]), whereas 'RR' indicates the recovery-robust solutions, for which D is part of the new objective. The value 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% for Palermo and between 18% and 49% for Genova, when $\pi = 2$. Figure 5.9 is a visualization of the data presented in Tables 5.1 and 5.4.

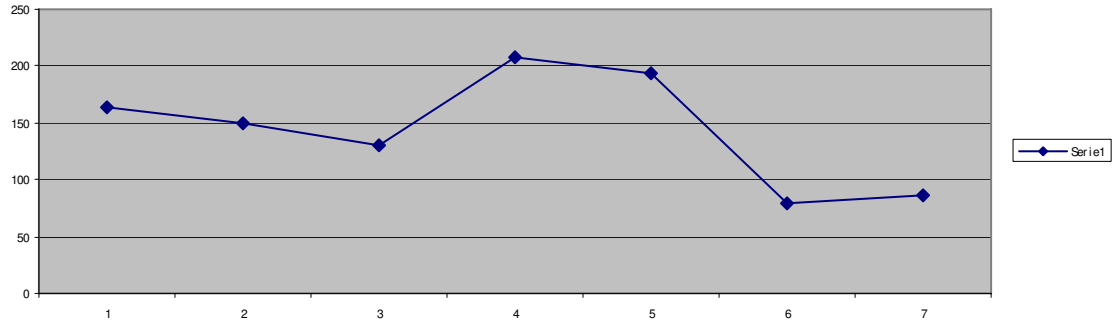

Figure 5.3: Graphic for Palermo Centrale, $\pi = 2$.

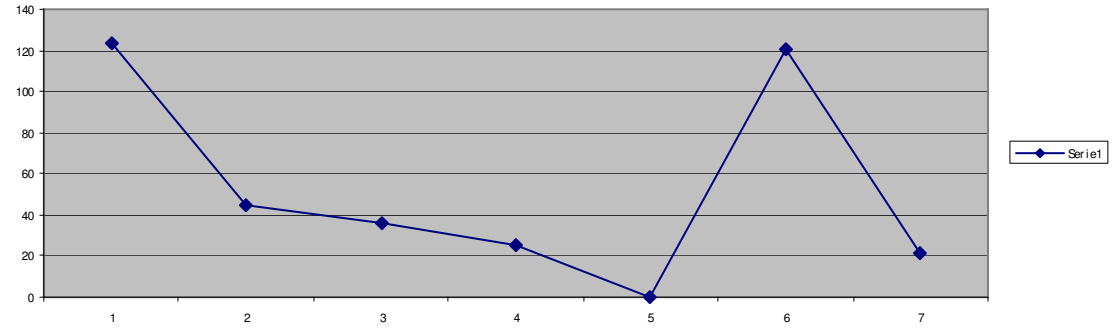

Figure 5.4: Graphic for Palermo Centrale, $\pi = 1$.

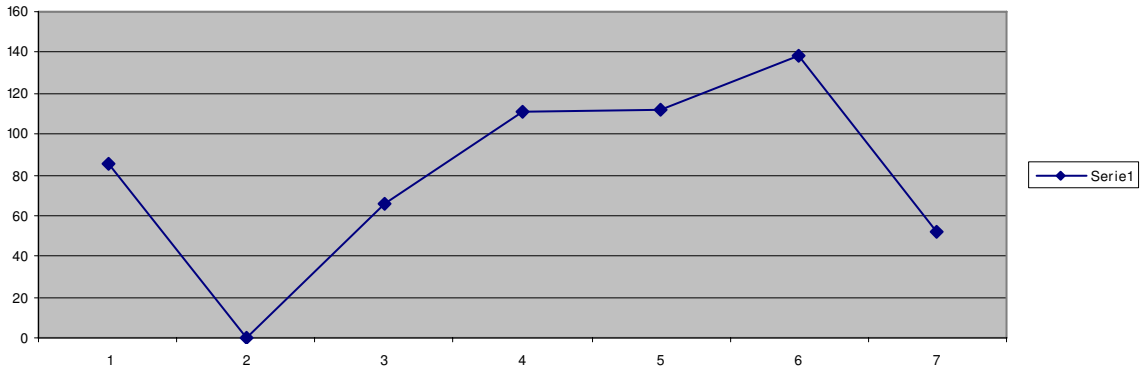

Figure 5.5: Graphic for Palermo Centrale, $\pi = 0$.

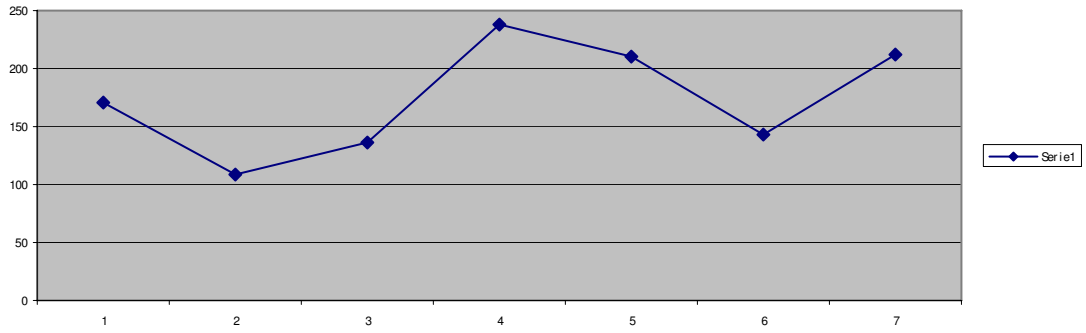

Figure 5.6: Graphic for Genova Piazza Principe, $\pi = 2$.

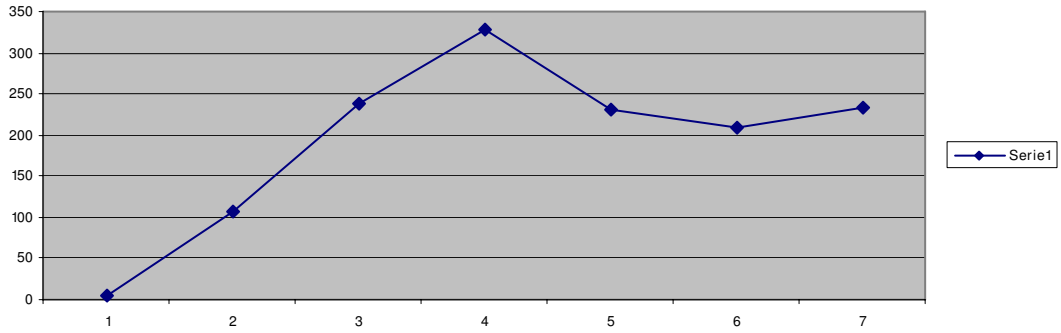

Figure 5.7: Graphic for Genova Piazza Principe, $\pi = 1$.

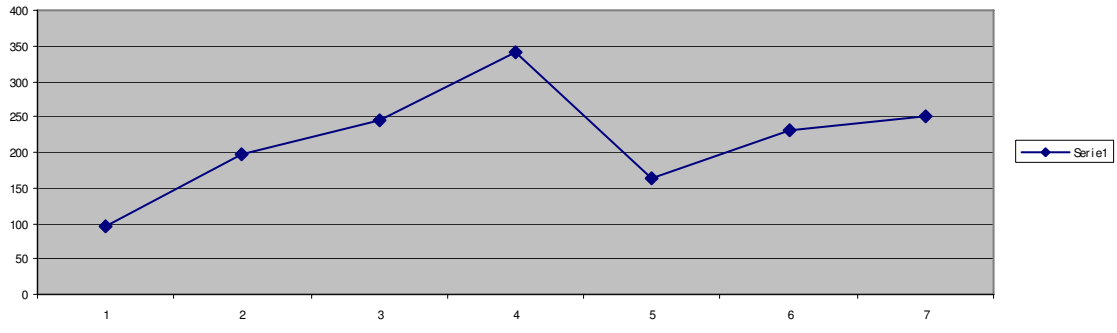

Figure 5.8: Graphic for Genova Piazza Principe, $\pi = 0$.

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 5.1: Results for Palermo Centrale, $\pi = 2$.

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	2	478	8	355	108	123	25.73
B: 07:30-09:00	5	563	7	518	4784	45	7.99
C: 09:00-11:00	3	460	6	424	198	36	7.83
D: 11:00-13:30	3	410	7	385	1246	25	6.10
E: 13:30-15:30	2	512	7	512	-	0	0.00
F: 15:30-18:00	3	495	6	374	15	121	24.44
G: 18:00-00:00	4	396	7	375	3144	21	5.30

Table 5.2: Results for Palermo Centrale, $\pi = 1$.

5.7 Conclusion

The network buffering approach allows to calculate robust train platforming plans for real-world instances. The buffering could be attached to an existing platforming software [11].

For all time windows and both train stations we considered the robust plans achieve nominal optimality, i.e., they assign the maximal number of trains that any platforming can assign in the instance. But they assign them in such a way that the maximal propagated delay in any scenario with less than Δ total seminal disturbance is, in some cases, even *halved* by the robust approach.

Note, that the deterministic method to which we compare the network buffering approach is not completely unaware of the tightness of the solutions. As mentioned it punishes the total time of (small) path conflicts. The computational study shows that this perspective is not sufficient to choose the most delay resistant among the nominally optimal platforming plans.

Thus, our method finds significantly more reliable solutions without extra costs.

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	2	574	8	489	533	85	14.81
B: 07:30-09:00	5	723	8	723	-	0	0.00
C: 09:00-11:00	3	563	7	497	23	66	11.72
D: 11:00-13:30	3	532	6	421	1006	111	20.86
E: 13:30-15:30	3	670	8	558	1039	112	16.72
F: 15:30-18:00	3	592	6	454	18	138	23.31
G: 18:00-00:00	4	548	9	496	57	52	9.49

Table 5.3: Results for Palermo Centrale, $\pi = 0$.

time window	# trains not platformed	D nom	CPU time nom (sec)	D RR	CPU time RR (sec)	Diff. D	Diff. D in %
A: 00:00-06:00	0	630	9	516	18190	114	18,10
B: 06:00-07:00	0	838	11	624	3177	214	25.54
C: 07:00-08:00	0	888	7	509	2495	379	42.68
D: 08:00-09:00	4	895	8	657	9940	238	26.59
E: 09:00-10:00	1	616	5	405	37	211	34.25
F: 10:00-11:00	1	516	5	373	14	143	27.71
G: 11:00-12:00	0	431	5	219	8	212	49.19

Table 5.4: Results for Genova Piazza Principe, $\pi = 2$

time window	# trains not platformed	D nom	CPU time nom (sec)	D RR	CPU time RR (sec)	Diff. D	Diff. D in %
A: 00:00-06:00	0	659	7	654	2617	5	0,76
B: 06:00-07:00	1	723	11	616	2719	107	14.80
C: 07:00-08:00	4	744	6	507	165	237	31.85
D: 08:00-09:00	4	906	13	579	8427	327	36.09
E: 09:00-10:00	1	707	6	475	23	232	32.81
F: 10:00-11:00	2	475	6	265	8	210	44.21
G: 11:00-12:00	1	449	5	216	8	233	51.89

Table 5.5: Results for Genova Piazza Principe, $\pi = 1$

time window	# trains not platformed	D nom	CPU time nom (sec)	D RR	CPU time RR (sec)	Diff. D	Diff. D in %
A: 00:00-06:00	1	717	8	621	73	96	13.39
B: 06:00-07:00	2	817	7	621	2829	196	23.99
C: 07:00-08:00	4	795	6	550	31	245	30.82
D: 08:00-09:00	5	960	7	620	13182	340	35.42
E: 09:00-10:00	3	518	7	355	17	163	31.47
F: 10:00-11:00	2	507	5	276	8	231	45.56
G: 11:00-12:00	1	481	5	231	7	250	51.98

Table 5.6: Results for Genova Piazza Principe, $\pi = 0$

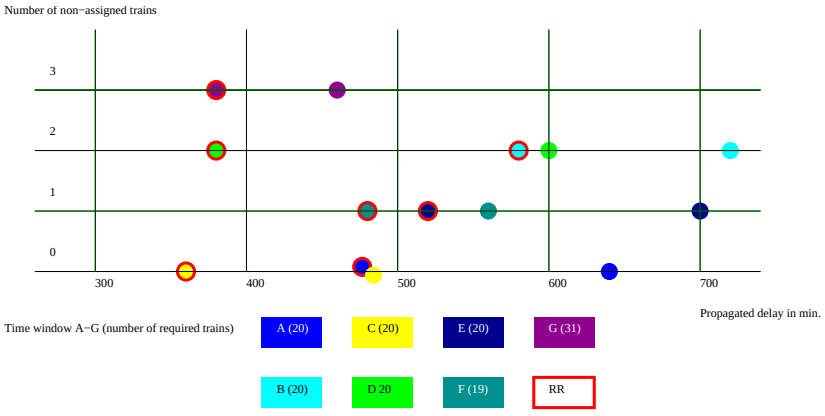

Figure 5.9: Delay propagation and through-put of robust and non-robust platforming.

Chapter 6

Robust Train Platforming and Online Rescheduling

As seen in the previous chapters, Train Platforming is a problem that arises in the early phase of the passenger railway planning process, usually several months before operating the trains. The main goal of platforming is to assign each train a stopping platform and the corresponding arrival/departure routes through a railway node. However, train delays often disrupt the routing schedules, thereby railway nodes are responsible for a large part of the delay propagation. In an ongoing research, with the computational tests still to be done, we propose robust models and rescheduling algorithms for computing robust routing schedules, and design a simulation framework to evaluate their robustness.

6.1 Introduction

Train platforming is a problem that arises in the early phase of the passenger railway planning process, usually several months before operating the trains. The main goal of train platforming is to route the trains through a railway node, for example through a busy station. Some research has been conducted in this field. For example, [35], [37], [6], [13], [14], [11] and [23] deal with the train platforming problem. These papers strictly focus on platforming as a planning problem. There is, however, very little literature available on robustness and re-scheduling issues for train platforming. [10] developed a model for finding delay-tolerant train routes. [15] consider a platforming in a re-scheduling setting. [26] describe a model to minimise the number of crossing train routes (and the time overlap thereof).

The goal of this work is threefold. The first goal is to extend the original platforming model of [11] to robustness considerations, describing some extensions where robustness is enhanced either by increasing the delay absorption capacity or by explicitly providing potential recovery possibilities. We formulate three versions of the original model. The first variant simplifies the original formulation: we do not penalise the overlapping among incompatible paths, i.e. only hard constraints on platform occupation are considered. In the second variant, the robustness is captured by penalising the occasions when an infrastructure element (e.g. a platform or a path) is utilised by two trains in a short succession. That is, the load on the infrastructure is spread in time and in space. The third variant is recovery-driven, too. The model assigns to each train its "primary platform" as well as a "back-up platform". The primary platform is announced for the passengers, and is to be used unless delays occur. Delayed trains may

be diverted to their back-up platform, again, provided that those are free. The key feature of the model is that a platform can serve as a back-up for a train t if no other train uses it as a primary platform in a time interval around the planned platform occupation of train t.

The second goal is to design different rescheduling algorithms for platforming and routing, based on the previous extended models. In particular, we consider three different rescheduling policies, implemented by an exact optimiser: the first consists in simply propagating the delay, the second relies on robust extra-resources (back-up platforms), the third allows all possible changes to the original plan.

The third goal is to assess the performance of the previous robust plans and rescheduling policies in a simulation framework. In fact, robustness and recoverability (rescheduling) are intriguingly difficult notions to quantify. Optimisation frameworks such as stochastic programming (see e.g. [7]) or robust optimisation (see [5]) have been developed to deal with the uncertainty of information and with the stochastic character of the problems. In this research we do not embark on defining and computing the "most robust" solutions to the train platforming problem. Instead, we propose a simulation framework to compare different schedules. More specifically, given a timetable of an entire day, a routing plan and some randomly-generated delays, the recovery algorithms are applied to resolve the routing conflicts. In this way, the simulation allows us to compare the robustness of different platforming and routing plans together with different rescheduling policies, the main criterion in the comparison being the sum of all train delays.

With the computational tests still to be done, we are confident that the simulation results shall give sufficient insight to analyse the behavior of the solutions to the three robust variants of the platforming model.

This chapter is organised as follows. In Section 6.2 we sketch the basic model of [11]. In Section 6.3 we describe the variants of the original model and in Section 6.4 we present three different rescheduling policies, implemented via ILP. We propose a simulation framework in Section 6.5, and discuss some of its crucial elements.

6.2 The Original Platforming Model

Here we recollect the original platforming model presented (and solved via ILP) in Chapter 2. The problem, as described by the Italian Infrastructure Manager, aims at defining a routing plan for a particular railway station, after the corresponding timetable has been defined. We are given a timetable containing all the trains that will either stop or travel through the railway station we are considering. The timetable, for every train, defines arrival and departure directions together with arrival and departure times. Times can be slightly moved according to some range associated to every train in the schedule. Information on the railway station topology is also given. A railway station can be represented as a node that connects different railway lines (or corridors). Inside the railway station, there are deposit areas (generally used for shunting operations), stopping platforms, and paths, i.e. sequences of tracks. Each corridor (external to the railway station) or deposit (internal area) is considered as a direction. Paths connect a given direction to a stopping platform. Different paths can share the same track or other physical resources along their lines, in this case they are considered incompatible. Finally, the constraints on the platforming operations consist of both hard and soft constraints. Hard constraints forbid the assignment of the same platform to different trains at the same time. Soft constraints are meant to penalise

situations that, while still allowing operations, may cause delays. More specifically, if two trains travel on incompatible paths at the same time, one of the two might have to be delayed to let the other go. Still, if the time windows, corresponding to the path occupation, overlap by a few minutes, the delay will not be crucial. For this reason, the Infrastructure Manager defines a maximum overlapping threshold, beyond which constraints become hard. Under the threshold, overlappings are penalised in the objective function, but are allowed. The goal is to find a platforming plan, i.e. define for every train in the timetable a stopping platform and two paths, connecting the platform to the arrival and departure directions of the given train. As said, the timetable allows small changes on the scheduled times. Hence, the platforming plan can also define the new arrival and departure times for every train. In this way, the routing phase feeds back to the previous planning phase (timetabling), where node capacities were not considered. Different iterations of the two phases usually take place, until a final plan is devised via subsequent refinements, as in the tradition of railway planners.

For this purpose, we defined the concept of pattern. A pattern encapsulate in a very compact form a complete assignment for a given train: it is a 5-tuple containing the information on the stopping platform, arrival and departure paths and times.

For a complete reference, see Chapter 2. Here we will simply report the complete original model.

$$\min \sum_{b \in B} c_b y_b + \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{(t_1,t_2) \in T^2} \sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ x_{t_1,P_1}\ x_{t_2,P_2} \tag{6.1}$$

subject to

$$\sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{6.2}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq y_b, \qquad K \in \mathcal{K}_b, \tag{6.3}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \tag{6.4}$$

$$y_b, x_{t,P} \in \{0,1\}, \qquad b \in B,\ t \in T,\ P \in \mathcal{P}_t. \tag{6.5}$$

The objective function (6.1) minimises the number of platforms used, the total cost of the selected patterns (the cost of a pattern represents the quality of the corresponding assignment for the given train, i.e. preference platforms, changes w.r.t. the nominal scheduled times of arrival and departure, etc.). Finally, the quadratic term models cross penalties, i.e. penalties that depend on the choice of two patterns simultaneously. Cross penalties are necessary in order to penalise occurrences of path conflicts for a train pair. In Chapter 2, we showed how to linearise the quadratic term, in order to obtain an ILP formulation for our problem. We also illustrated how to deal smoothly with the separation of clique constraints (6.3) and (6.4). The same arguments apply for all the variants that we will present in the sequel. Thus, for every detail on how to solve the original ILP of train platforming, the reader can refer to the corresponding chapter.

6.3 Robust Planning

Here we present three different variants of the model of Caprara et al. [11].

6.3.1 Basic Platforming

Model In the first variant, called *basic*, we simplify the original model, by considering exclusively the cost of the patterns, without any additional fixed cost for the platforms used. In fact, in a robust model it may be desirable for the trains to spread platform occupation among different resources. Even penalties for path conflicts are ignored. Only hard constraints are considered, i.e. we do not penalise overlappings on incompatible paths, but we simply forbid those that are over the threshold, together with any platform conflict. The model of Section 6.2 becomes:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{t \in T} M_t s_t \tag{6.6}$$

subject to

$$s_t + \sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{6.7}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \tag{6.8}$$

$$x_{t,P} \in \{0,1\}, \qquad t \in T,\ P \in \mathcal{P}_t. \tag{6.9}$$

Notice that constraints (6.3) are not needed, because we do not have to link platform variables with pattern variables anymore. Platform conflicts are all included in (6.8) together with over-threshold path conflicts. Variables s_t are slacks.

Solution method As explained in Chapter 2, *dummy platforms* were introduced in order to measure the lack of capacity for the station considered. Moreover, in order to carry out a solution approach based on a *branch-and-cut-and-price* framework, it was necessary to initialize the master problem with a set of variables corresponding to a feasible solution. Thus, a fast, greedy algorithm was run first, and the presence of dummy platforms was crucial to make a feasible solution *always* available. Yet, in a robust-oriented application, we are not interested in capacity issues, since resource saving may adversely affect the robustness of the solution. Hence, the basic model for robust planning only considers *real* platforms. On the other hand, since we intend to apply the original solution approach also in this case, we need a way to initialize the master with a feasible solution. This cannot be done via heuristic algorithm, because the feasibility version of the problem is *NP-Hard*. For this reason, in order to enforce feasibility throughout the whole algorithm, we introduced some slack variables s_t for the covering constraints (6.7). These variables have a very high cost M_t, such that none of them will ever be chosen if a feasible platforming exists. The master is initialized with the slack variables s_t only, and the covering constraints (6.7). During the pricing phase, $x_{t,P}$ variables with negative reduced cost are added and the corresponding constraints are updated. Constraints (6.8) are separated considering a weaker formulation, i.e. we consider incompatibility graphs for every pair of trains (see Chapter 2 for more details).

6.3.2 Increasing the Absorption Capacity

The most straightforward way to cope with small delays without any particular re-routing strategy is to simply propagate the delay. The *delay propagation* recovery policy consists in

resolving the given scenario by preserving the original scheduled train order. For this reason, it is important to increase the delay absorption capacity of the routing plan.

If a train is delayed then all the tracks that it was supposed to be occupying are locked and the subsequent trains, if allocated to one or more of these, are pushed-back, waiting for the corresponding resource to be freed up.

Model Even though the quadratic component contained in the original model and described in Section 6.2 was designed to weight the number of minutes of path conflicts generated by a particular choice of patterns for each train pair, this formulation proves to be particularly flexible. Indeed, through the quadratic term, one could weight any cross effect among pairs of patterns. For example, we could model some kind of robustness by penalising incompatible resources that are used too close in time and thus may affect feasibility under small disturbances. In fact, in the second variant of robust planning, called *Increasing Absorption Capacity*, the robustness is captured by penalising the cases in which an infrastructure element (a platform or a route) is utilized by two trains in a short succession. These may give rise to resource conflicts in case of train delays and thereby may lead to propagation of such delays. So the aim is to spread the load on the infrastructure in time and space. A routing plan optimised for this criterion is expected to cope with small, accumulating delays, without substantial recovery actions.

$$\min \sum_{t\in T}\sum_{P\in\mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{t\in T} M_t s_t + \sum_{(t_1,t_2)\in T^2}\sum_{P_1\in\mathcal{P}_{t_1}}\sum_{P_2\in\mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ x_{t_1,P_1}\ x_{t_2,P_2} \quad (6.10)$$

subject to

$$s_t + \sum_{P\in\mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \quad (6.11)$$

$$\sum_{(t,P)\in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \quad (6.12)$$

$$x_{t,P} \in \{0,1\}, \qquad t \in T,\ P \in \mathcal{P}_t. \quad (6.13)$$

The cross-penalty c_{t_1,P_1,t_2,P_2} in the objective function is a *compound* cost defined by summing up 5 different costs that depend on the pair of patterns considered.

In Figure 6.1 we illustrate the resource occupation for a pair of trains. Both segments (the upper and the lower one) consist of three sub-segments representing the occupation time windows for the three resources that are assigned to every train: arrival path (pink), stopping platform (blue) and departure path (green). For each pair of time windows, corresponding to possibly incompatible resources, we have to calculate the distance and define a cost according to it:

1. $c^{aa}_{t_1,t_2}$: cost-distance function for arrival path of t_1 *vs* arrival path of t_2
2. $c^{ad}_{t_1,t_2}$: cost-distance function for arrival path of t_1 *vs* departure path of t_2
3. $c^{da}_{t_1,t_2}$: cost-distance function for departure path of t_1 *vs* arrival path of t_2
4. $c^{dd}_{t_1,t_2}$: cost-distance function for departure path of t_1 *vs* departure path of t_2
5. $c^{ss}_{t_1,t_2}$: cost-distance function for stopping platform of t_1 *vs* stopping platform of t_2

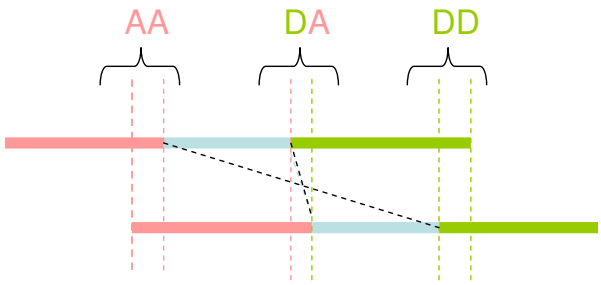

Figure 6.1: Resource occupation time windows.

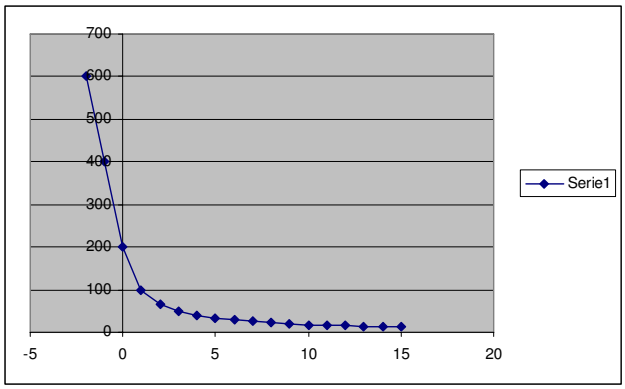

Figure 6.2: Path-distance cost function.

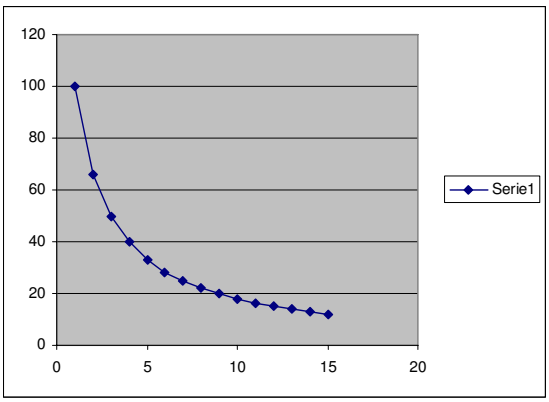

Figure 6.3: Platform-distance cost function.

We define a maximum distance of 15 minutes, above which, the cost is set to 0. The costs are piecewise monotone decreasing functions of the distance. More precisely, for the path-incompatibility costs, functions are defined for distances ranging from 15 minutes to -π. In fact, path incompatibilities allow overlapping under the threshold π, see Figure 6.2. Whereas for platform incompatibilities, we consider distances ranging from 1 to 15 minutes, because overlapping is forbidden on platform resources, see Figure 6.3.

Even though the quadratic objective can express a wide array of optimisation criteria, each of them would require a fine-tuning of the coefficients in the objective function. A common way to perform such a tuning is to apply scenario-based models such as stochastic programming. Still, the performance of the overall method would strongly rely on the chosen distribution and on the absorption capacity of the network, possibly leading to an inefficient as well as practically non-robust routing plan. The main drawback of this approach is represented by the complete lack of information on the delay distribution and thus on the convenience of a particular tuning with respect to another.

Moreover this formulation, though extremely versatile, is not suited to define the amount of robustness we are willing to attain nor to measure the cost of it. For this reason we set out to apply a more deterministic approach, presented in the next section.

Solution method The quadratic term in (6.10) is linearised via the technique described in Chapter 2. Linearisation constraints are separated together with clique constraints (6.12), and pattern variables are generated dynamically, by enumerating all the patterns and adding, for each train, the one with minimum negative reduced cost.

6.3.3 Back-up Platforms

Robustness, more precisely Recoverable Robustness, can be interpreted as the possibility to perform easy recovery actions under a given set of delay scenarios. The level of robustness consists in the number of scenarios we are able to deal with, and the *price of robustness* is represented by the resources dedicated to recovery purposes, i.e. the loss in optimality. In this third version, we intend to fight delay propagation by allowing each train to use a recovery or *back-platform* for a given time period, whose length coincides with the maximum delay we intend to prevent. The original model objective function included a fixed cost for each platform used. The goal of keeping as many tracks free as possible, can be interpreted as some kind of recovery purpose. In fact, it may be not a thriving technique, but it is certainly tempting to keep one or more tracks free to be used exclusively for recovery actions. The recovery algorithm could work as follows: whenever a train is delayed we keep the same scheduling and allocation as long as possible, that is until propagation would occur. Only at this point, we turn to a recovery platform and divert the delayed train to one of the free tracks. Of course if no free tracks are available, propagation will be inevitable. As said, the idea of free extra tracks was already incorporated in the original model by [11], although in that context their purpose was to handle cases in which the capacity of the railway station was not sufficient to deal with the service request and they were thus meant to measure the lack of capacity. This approach (*reserve platforms*) is, though fairly simple to be designed, practically difficult to be realized. First of all, it is quite unlikely that an available platform will be kept free exclusively for recovery purposes. Moreover the topology of the railway station must be taken into account as not all platforms can be reached from any direction,

in fact there may not exist a path linking a given pair of directions with the platform we are given.

Hence in the third variant, we designed a fairly more complex recovery policy. We assign each train a "primary platform" as well as a "back-up platform". The primary platform is announced to the passengers and has to be used unless delays occur. Delayed trains may be diverted to their back-up platform allowing re-scheduling possibilities. The concept of back-up platforms differs from reserve platforms because the spare platform capacity is spread in space and time rather than focused on a few single tracks all day long. The compelling need of free recovery resources, led us to figure out a trade-off between a completely free platform and the mere delay propagation policy. The solution deviced consists in spreading the reserve capacity among different platforms, allowing each train to have a back-up track available for a given amount of time. The longer the back-up resource is available the larger delay we are able to fight.

The approach explained below is motivated by the observation that platform availability appears to be the most binding constraint in real-life passenger train routing operations.

Model In this model, each train is assigned a *primary* and a *backup* platform. A platform can only be backup for a train if no other train uses it as primary platform. The primary platform is intended to be used whenever possible, while the backup platform provides re-scheduling possibilities when the train is delayed. Suppose that train t has arrival time a and departure time d, and let ℓ be a limit on delays that we are dealing with. Then train t occupies its primary and backup platform during the time intervals $[a; d]$ and $[a; d + \ell]$, respectively. Note that the backup platform may be occupied if another train is delayed and uses it either as primary or as backup.

Platforms that have nearly identical approach routes are called *neighboring platforms*. Note that the precise definition depends on the infrastructure of the studied stations. We suggest to choose neighboring primary and backup platforms for each train. The reason for this is that, if a train is moved to a neighboring platform, it is very likely to use paths with the same structure as the original one, thus will not introduce many different conflicts with other routes already assigned. Moreover, passengers will be less disappointed if asked to move to a close by platform.

Backup platforms can be incorporated into the model (6.1)-(6.5) simply by extending the notion of patterns. However, such an extension would significantly increase the complexity of the basic routing model. This motivates another approach.

Define a set of intervals I_t (one for each train) corresponding to the time windows during which the backup platform for train t must be available and use $u_{\pi,t}$ binary variables that take value 1 if train t can use backup platform π in its corresponding time interval I_t. With this in mind we require for each train the assignment of a platform and a backup one in the neighborhood of the primary assignment:

$$\sum_{\pi \in N(b)} u_{\pi,t} \geq \sum_{P \in P_b} x_{t,P}, \qquad t \in T, b \in B, \tag{6.14}$$

where $N(b)$ is the set of neighboring platforms of b and P_b is the set of all the pattern that use b as primary platform.

Of course, we also require the backup platform to be free from any primary assignment in the desired time interval I_t

$$\sum_{t' \in T, P \in \mathcal{P}_{t'} : B(P)=\pi, I_t \cap [a^P_{t'}, d^P_{t'}] \neq \emptyset} x_{t',P} \leq M(1 - u_{\pi,t}), \qquad \pi \in B, t \in T. \tag{6.15}$$

The right hand side sums up all the patterns of all trains $t' \in T$, such that π is their primary platform ($B(P) = \pi$) and their occupation time interval $[a^P_{t'}, d^P_{t'}]$ intersects I_t. The main disadvantage in this formulation is the "big M" notation, although "M" may turn out to be small in practice. An alternative formulation considers for each platform all the possible arrival instants $m \in I_t$. Since at each instant only one train can use a given platform, this definition allows us to formulate constraints (6.15) without the "M", indeed the right hand side in (6.15) would be at most 1. More precisely the alternative formulation for constraints (6.15) is the following:

$$\sum_{t' \in T, P \in \mathcal{P}_{t'} : B(P)=\pi, m \in [a^P_{t'}, d^P_{t'}]} x_{t',P} \leq 1 - u_{\pi,t}, \qquad \pi \in B, t \in T, m \in I_t. \tag{6.16}$$

Hence, the overall model reads as follows:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{t \in T} M_t s_t \tag{6.17}$$

subject to

$$s_t + \sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{6.18}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \tag{6.19}$$

$$\sum_{\pi \in N(b)} u_{\pi,t} \geq \sum_{P \in P_b} x_{t,P}, \qquad t \in T, b \in B, \tag{6.20}$$

$$\sum_{t' \in T, P \in \mathcal{P}_{t'} : B(P)=\pi, m \in [a^P_{t'}, d^P_{t'}]} x_{t',P} \leq 1 - u_{\pi,t}, \qquad \pi \in B, t \in T, m \in I_t, \tag{6.21}$$

$$x_{t,P} \in \{0, 1\}, \qquad t \in T,\ P \in \mathcal{P}_t. \tag{6.22}$$

Solution method The solution approach to (6.17)-(6.22) is similar to the one used for the two previous versions. Yet, in this case we need to take care of two more types of constraints: (6.20) and (6.21). Both are inserted in the initial master, together with backup platform variables $u_{\pi,t}$. Thus, we only generate pattern variables $x_{t,p}$ and clique constraints (6.19). Notice that constraints (6.20), in the very first iteration, contain exclusively backup variables. Later, during the following iterations, when new patterns are inserted in the model, these constraints will be updated accordingly. Similarly, constraints (6.21) are inserted in the initial master program, containing only backup variables. Then, they will be updated according to the pattern generation process. Constraints updating is necessary also to carry out the pricing problem. In fact, all

patterns are enumerated and for each of them the reduced cost is calculated by considering all the constraints (currently in the model) that the new variable would affect, if inserted. Hence, pricing and updating can be done simultaneously and the overall process is not too demanding in terms of time of computation. In fact, we use an efficient indexing structure to handle both types of constraints (both in the pricing and updating phases).

6.4 Online Rescheduling

Once a (robust) routing plan is given, one has to test its effectiveness against delays by applying to it a recovery strategy, which may be a general one or it may be designed starting from the specific robust plan.

6.4.1 Recovery strategies

Here we deal with three different recovery strategies.

Delay Propagation This is a general strategy in which each train keeps its originally-assigned route and the order of trains on the trajectories is not modified either. Thus conflicts are resolved by adjusting the arrival and departure times and by eventually propagating the delay. This strategy can be applied to any of the plans described in the previous section.

Using Back-up Platforms This strategy is applicable when back-up platforms are available, and assigns trains either to their planned platform or to their back-up platform, whichever is available earlier. The associated rescheduling algorithm tries to exploit the recovery resources provided by the plan, in order to minimise the delay propagation. This specific strategy is only applicable for the back-up robust type of plan, since we need specific information on the recovery resources.

Full Recovery This general recovery rule is the most powerful one. It allows any kind of rescheduling action. Hence, trains may be assigned to completely different patterns with respect to the original plan. Being a general strategy, it can be applied to all types of plans.

6.4.2 Rescheduling algorithms

For each of these strategies, we implemented an *optimisation-based* rescheduling algorithm obtained by expressing mathematically the corresponding rules. This can be done extending the model (6.1)-(6.5). The rescheduling process is performed by solving the associated mathematical program.

As we will see in the next section, dedicated to the simulation framework, the rescheduling algorithm is run every minute and updates the current plan (which at the first iteration corresponds to the original robust plan) in order to cope with delays. Delays are simulated providing a new timetable, where the arrival times of the trains are not the nominal ones (i.e. those used in the robust planning phase), but *real-time* information on the *expected time of arrival* (ETA). This information is updated every minute and becomes more precise as time goes by. For this reason, we refer to *online* rescheduling. Each timetable-update feeds the

rescheduling algorithm, which proposes a new plan according it, trying to resolve conflicts using a specific recovery policy.

Freeloading from the concept of pattern, which allows a flexible definition of the assignment possibilities, each recovery rule is implemented defining for each train a proper set of feasible patterns, according to the recovery actions allowed by the corresponding policy.

Before giving details on each of these strategies, we will illustrate some features that are common to all optimisation-based rescheduling models.

Delay propagation

The most simple recovery action, which all three policies include, consist in delaying a train. This can be represented mathematically as a pattern with a shift forward on the arrival time of the train, whose value corresponds to the delay we want to impose. As explained in Chapter 2, the original formulation for platforming already modelled the possibility of shifts forward and backward, with respect to the nominal schedule. Similarly, the rescheduling algorithms can use shifts to delay trains, in particular we use shifts forward, with a maximum length of 20 minutes.

The objective function

The objective function of all rescheduling algorithms aims at minimising the total propagated delay. First, we need to define the concept of *propagated delay*, since the possibility to change arrival and departure times via shift can affect delay propagation in three different ways. Figure 6.4 shows all the possible shifts that can be applied to a train. The expected arrival time (ETA), can be shifted forward (pink dotted line): we call this shift δ_1. The travel time on the arrival path a_t is constant (pink line). Each train has a minimum dwelling time d_t (blue line), thus the nominal departure time can be shifted backward (green-blue dotted line) or forward (green dotted line): we call these shifts respectively δ_2 and δ_3. Note that $\delta_2 \leq 0$.

There are at least three simple ways to weight the propagated delay Δ:

- $\Delta = \delta_1 + \delta_2 + \delta_3$.
- let $EPTD$ be the earliest possible time of departure: $EPTD = ETA + a_t + d_t$. Then, $\Delta =$ *realised departure time* $- EPTD$.
- $\Delta =$ *realised departure time - planned departure time.*

For each pair (t, P) (train-pattern), $\Delta_{t,P}$ expresses the delay propagated for train t, if it is assigned to pattern P. Hence, the objective function looks as follows:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} \Delta_{t,P} x_{t,P}. \tag{6.23}$$

Note that a major simplifying factor in (6.23) is the lack of the quadratic term. The total propagated delay is minimised and this objective is a linear function in terms of the $x_{t,P}$ variables.

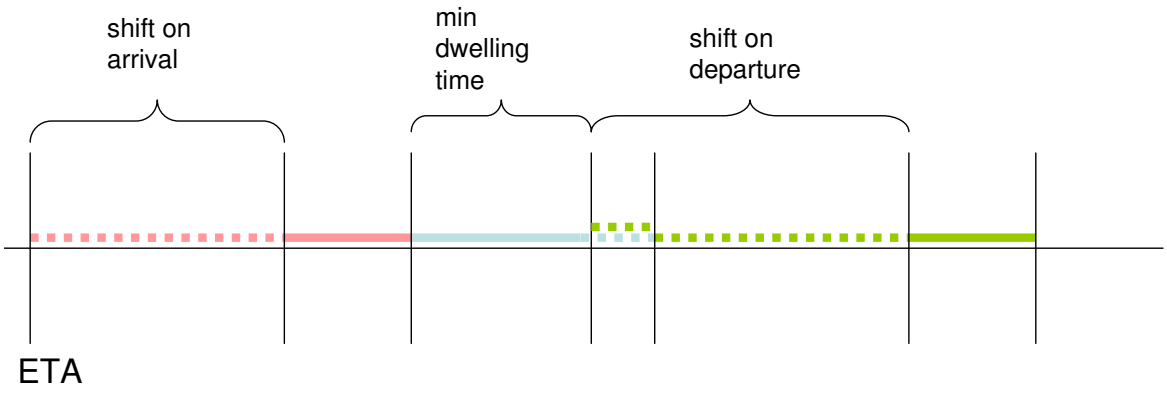

Figure 6.4: Possible shifts.

Shift constraints

As said, the IP models for the rescheduling algorithms maintain the original structure (6.1)-(6.5). On the other hand, these models must allow rather large additional arrival/departure shifts in order to be able to deal with train delays. Hence, the number of admissible patterns is much higher than in the planning applications. Further, the queues of trains on the in-bound tracks require additional constraints. In fact, as shown in Figure 6.5, trains that are delayed on arrival, queue up outside the railway station on some in-bound tracks. In the picture, we represent the railway station as a system of platforms (blue oval), arrival and departure paths (pink and green arrows respectively). Overtake among trains on arrival is not possible, hence we need to forbid shift values that correspond to these occasions. In other words, if $ETA_{t_1} < ETA_{t_2}$, we impose $ETA_{t_1} + \delta_1(t) < ETA_{t_2} + \delta_1(t)$. We can express this with constraints of the form:

$$\sum_{(t,p)\in S} x_{t,p} \leq 1. \tag{6.24}$$

These constraints simply forbid the simultaneous occurrence of certain sets S of patterns, if these decisions together indicate that a train joins the queue of waiting trains earlier but leaves it later.

Consider an example with three different trains, whose expected time of arrival are respectively $ETA(t_1)$ =10:00, $ETA(t_1)$ =10:02, $ETA(t_1)$ =10:05. For simplicity, suppose the maximum allowed shift forward on arrival (max δ_1) to be 5 minutes. Whenever we apply some shifts on these trains, we want to keep the original order corresponding to the ETA instants, i.e. in this case: $t_1 \prec t_2 \prec t_3$.

In Figure 6.6 each oval represents a train, with all the possible arrival times according to the different shift values. A red arrow from one node to another means that the corresponding two shift values are incompatible. This happens either when the shifts make the two arrival times equal or change the order. Note an important characteristic of this *shift incompatibility graph*: edges go from one train to the following one, but because of the transitive property, we do not need to specify those crossing one or more trains.

Each node within the oval represents all the patterns, for the given train, using that particular shift value. Thus, separation of constraints (6.24) can be done by looking for

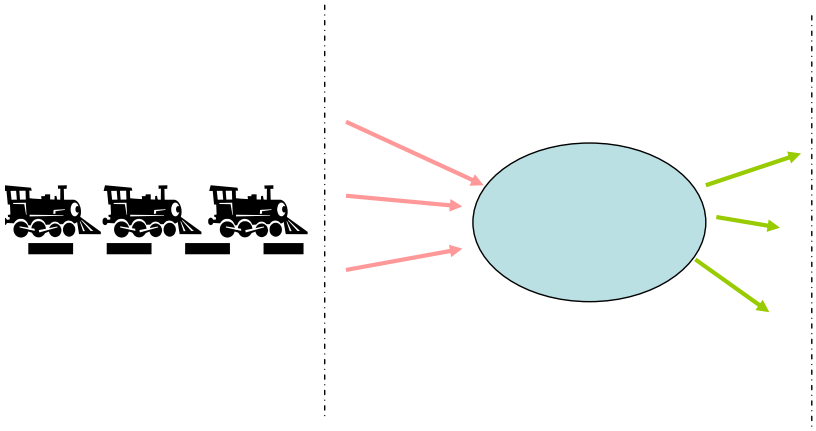

Figure 6.5: Train queue on arrival.

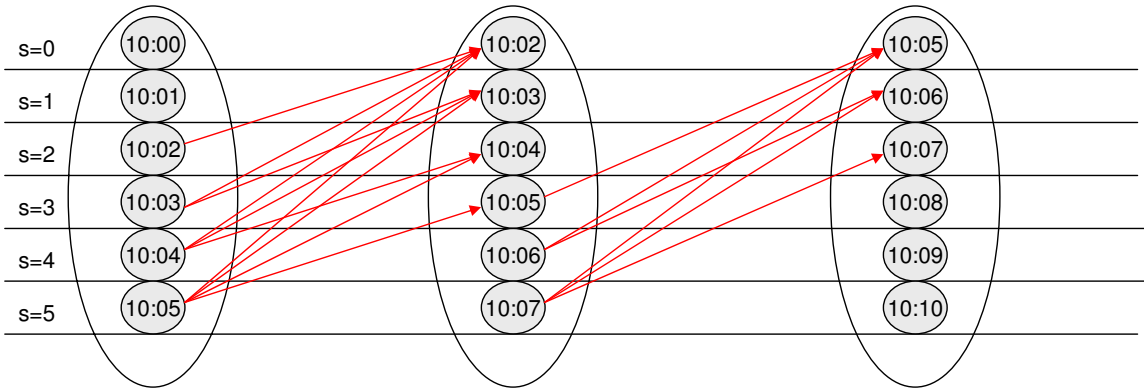

Figure 6.6: Shift incompatibility graph.

maximum weight cliques on the shift incompatibility graph.

Also note that instead of considering *nodes* (single shift values), we can consider *shift-intervals*, in order to make the formulation stronger. Each shift-interval contains subsequent shift values (gray nodes) and its weight corresponds to the sum of all the pattern variables of each node.

Finally observe that if we are given a set S of incompatible shift-intervals ι and we want to extend it in order to construct a clique, all the shift-intervals $\iota \notin S$ that are incompatible with our set (i.e. those entitled for the extension) depend exclusively on the single shift-interval in S of the last arriving train (the one with maximum ETA). Hence, when computing a maximum weight clique on the shift-incompatibility graph, we can consider every train as a *stage* and calculate the maximum profit for each shift-interval of the train by taking all the incompatible shift-intervals of the previous train. In other words, separation can be carried out via *dynamic programming*.

The overall rescheduling model reads as follows:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} \Delta_{t,P}\ x_{t,P} + \sum_{t \in T} M_t s_t \tag{6.25}$$

subject to

$$s_t + \sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{6.26}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \tag{6.27}$$

$$\sum_{(t,P) \in S} x_{t,P} \leqslant 1 \qquad S \in \mathcal{S}, \tag{6.28}$$

$$x_{t,P} \in \{0,1\}, \qquad t \in T,\ P \in \mathcal{P}_t. \tag{6.29}$$

Each of the recovery policies we have mentioned above is implemented by generating, for each train, all the patterns compatible with the recovery rule. This can be done easily, since the patterns are enumerated and we do not need any mathematical formulation to describe them.

Delay propagation recovery rule allows changes on the shift values, but the resources of the trains are fixed, according to the original plan.

Back-up platform recovery strategy also allows changes on the resources, but these changes are limited to the recovery resources, i.e. a train can be shifted and/or moved to the backup platform, and the approaching routes can be adapted according to it.

Full recovery policy allows any change on the original plan, i.e. all resources and times can be modified.

It is important to note that the set of possible patterns for each train depends not only on the recovery strategy we are implementing, but also depends on time. In particular, there are 4 different "types" of trains according to their ETA value, that the rescheduling algorithm has to consider, as shown in Figure 6.7. We said that each train has an expected time of arrival (ETA), defined by the simulation framework and updated every minute. Let t be the current time: the simulation framework feeds the recovery algorithm with the new information and asks for a new plan for the trains whose ETA falls within one hour from the current time. The other trains are ignored (since the corresponding information is not reliable enough yet). Moreover, trains that are expected within 15 minutes time can only be shifted, but the corresponding resources are fixed, since the passengers should not be forced to move from one place to another of the railway station. Trains that have already arrived, but have not left yet can be delayed on departure, but of course not on arrival and their resources are fixed. Finally, trains that have already left, but are still occupying the departure path have their patterns fixed.

Here we summarize the different types of trains according to their time of arrival and departure, and the corresponding recovery possibilities.

1. ETA of type 0: these trains are expected to arrive within one hour, but not before 15 minutes. Hence their patterns can be generated completely according to the recovery strategy.

Figure 6.7: Different train types according to ETA.

2. ETA of type 1: these trains are expected to arrive within 15 minutes, thus their resources are fixed. The only possible changes regard the shifts, in particular the arrival and departure times can both be shifted.

3. ETA of type 2: these trains have already arrived, but have not left yet. Hence, we can only change the corresponding departure time via shift.

4. ETA of type 3: these trains have already left, but they are still occupying the departure path. We need to consider them in the rescheduling problem, since some resources are being blocked, but their patters are completely fixed.

6.5 Simulation Framework

The simulation framework is used to assess the level of robustness of a routing plan and the performance of a recovery algorithm under realistic train delays. The input of the simulation framework consists in a routing plan, a delay distribution and a recovery algorithm. The simulation involves all train arrivals and departures at a single railway node on a given day. In-bound trains are subject to random delays according to some realistic probability distribution.

The recovery process takes place at every minute and works over the subset of trains arriving in the next hour time span. Each time the recovery process has to decide on assigning new resources to trains and on modifying their arrival and departure times. This simulation technique is referred to as *rolling horizon*, since the re-optimisation concerns a 1-hour horizon which rolls forward minute after minute. It is assumed that decisions for the forthcoming 15 minutes are announced to the passengers and cannot be modified. Decision for later events, however, may be revoked as the rolling horizon shifts further. The simulation framework can take any of the robust routing solutions, and it can be run with any of the aforementioned recovery strategies. This provides insights to which extent are the recovery algorithms able to cope with train delays. In particular, one can analyze the benefit of using the optimisation based recovery algorithm compared to simple rule-based heuristics. Also, one can compare the recoverability potential of different routing plans.

Bibliography

[1] Ben-Tal A., El Ghaoui L., Nemirovski A.: *Special issue on robust optimization.* Mathematical Programming A, **107(1-2)** (2006).

[2] Ben-Tal A., Nemirovski A.: *Robust Convex Optimization.* Mathematics of Operations Research, **23** (1998) 769-805.

[3] Ben-Tal A., Nemirovski A.: *Robust solutions to uncertain programs.* Operations Research Letters, **25** (1999) 1-13.

[4] Ben-Tal A., Nemirovski A.: *Robust solutions of linear programming problems contaminated with unceratin data.* Mathematical Programming, **88** (2000) 411-424.

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

[6] Billionnet A.: *Using Integer Programming to Solve the Train Platforming Problem.* Transportation Science, **37** (2003) 213-222.

[7] Birge J., Louveaux F.: *Introduction to Stochastic Programming.* Springer, New York, (1997).

[8] Bonomo F., Durán G., Marenco J.: *Exploring the Complexity Boundary between Coloring and List-Coloring.* Electronic Notes in Discrete Mathematics, **25** (2006) 41-47.

[9] Boyd S., Ben-Tal A., Nemirovski A.: *Extending scope of robust optimization: Comprehensive robust counterparts of uncertain problems.* Mathematical Programming (2006).

[10] Caimi G., Burkolter D., Herrmann, T.: *Finding Delay-Tolerant Train Routings through Stations.* Operations Research Proceedings 2004, Fleuren H., (2007), 136-143, Springer.

[11] Caprara A., Galli L., Toth P.: *Solution to the Train Platforming Problem.* Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS), (2007).

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

[13] Carey M., Carville S.: *Scheduling and Platforming Trains at Busy Complex Stations.* Transportation Research A, **37** (2003) 195-224.

[14] Carey M., Crawford I.: *Scheduling Trains on a Network of Busy Complex Stations.* Transportation Research B, **41(2)** (2007) 159-178.

[15] Chakroborty P., Vikram D.: *Optimum Assignment of Trains to Platforms under Partial Schedule Compliance.* Transportation Research B, **(42:2)** (2008) 169-184.

[16] Cicerone S., D'Angelo G., Di Stefano G., Frigioni D., Navarra A.: *Robust Algorithms and Price of Robustness in Shunting Problems.* Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS), (2007).

[17] De Luca Cardillo D., Mione N.: *k L-List T Colouring of Graphs.* European Journal of Operational Research, **106** (1999) 160-164.

[18] El-Ghaoui L., Lebret H.: *Robust solutions to least-square problems to unceratain data matrices.* SIAM J. Matrix Anal. Appl., **18** (1997) 1035-1064.

[19] El-Ghaoui L., Oustry F., Lebret H.: *Robust solutions to unceratin semidefinite programs.* SIAM J. Optim., **9** (1998) 33-52

[20] Fioole P., Kroon L., Maróti G., Schrijver A.: *A rolling stock circulation model for combining and splitting of passenger trains.* European Journal of Operational Research, **174** (2006) 1281-1297.

[21] Fischetti M., Monaci M.: *Light Robusteness* Techinical Report 0119, EU ARRIVAL project.

[22] Fishburn P.C.: *Interval Orders and Interval Graphs–A Study of Partially Ordered Sets.* John Wiley, New York, (1985).

[23] Fuchsberger M.: *Solving the Train Scheduling Problem in a Main Station Area via a Resource Constrained Space-Time Integer Multicommodity Flow.* Institute for Operations Research, ETH Zürich, Switzerland, (2007).

[24] Garey M.R., Johnson D.S., Miller G.L., Papadimitriou C.H.: *The complexity of coloring circular arcs and chords.* SIAM J. Algebraic Discrete Methods, **1**, (1980) 216–227.

[25] Goldberg A.V., Tarjan R.E.: *A New Approach to the Maximum Flow Problem*, Proceedings of the 18th ACM Symposium on the Theory of Computing (1986).

[26] Kroon L.G., Maróti G.,: *Robust Train Routing.* Techinical Report 0123, EU ARRIVAL project.

[27] Kroon L.G., Romeijn H.E., Zwaneveld P.J.: *Routing Trains Through Railway Stations: Complexity Issues.* European Journal of Operations Research, **98** (1997) 485-498.

[28] Liebchen C., Lübbecke M., Möhring R. H., Stiller S.: *Recoverable Robusteness.* Techinical Report 0066, EU ARRIVAL project.

[29] Liebchen C., Stiller S.: *Delay Resistant Timetabling.* Techinical Report 0066, EU ARRIVAL project.

[30] Maróti G.: *Operations Research Models for Railway Rolling Stock Planning.* Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, (2006).

[31] Nemhauser G., Wolsey L.: *Integer and Combinatorial Optimization.* Wiley, (1988).

[32] Nielsen L.: *Disruption Generator for Railway Rolling Stock Re-scheduling.* Software, (2008).

[33] Soyster A.L.: *Convex Programming with set-inclusive constraints and applications to inexact linear programming.* Operations Research **(21:4)** (1973) 1154-1157.

[34] Stiller S.: *Extending Concepts of Reliability.* Ph.D. Thesis, Institut für Mathematik, Technische Universität Berlin, Berlin, Germany, (2008).

[35] Zwaneveld P.J., Kroon L.G., Romeijn H.E., Salomon M., Dauzere-Peres S., van Hoesel C.P.M., Ambergen H.W.: *Routing Trains Through Railway Stations: Model Formulation and Algorithm*, Transportation Science, **30** (1996) 181-194.

[36] Zwaneveld P.J.: *Railway Planning and Allocation of Passenger Lines*, Ph.D. Thesis, Rotterdam School of Management, Rotterdam, The Netherlands, (1997).

[37] Zwaneveld P.J., Kroon L.G., van Hoesel C.P.M.: *Routing Trains through a Railway Station based on a Node Packing Model.* European Journal of Operations Research, **128** (2001) 14-33.