Optimal Train Dispatching by Benders’–Like Reformulation

Article in Transportation Science · January 2015
DOI: 10.1287/trsc.2015.0605

3 authors:

Carlo Mannino
Sapienza University of Rome
58 PUBLICATIONS 1,234 CITATIONS
See profile

Leonardo Lamorgese
Optrail
12 PUBLICATIONS 32 CITATIONS
See profile

Mauro Piacentini
Sabre
11 PUBLICATIONS 45 CITATIONS
See profile

Some of the authors of this publication are also working on these related projects:

- SESAR WP-E ZefMap View project
- Solution of SBILP View project

All content following this page was uploaded by Carlo Mannino on 01 March 2017.
The user has requested enhancement of the downloaded file.
Optimal Train Dispatching by Benders’-like reformulation

Leonardo Lamorgese *  Carlo Mannino *  Mauro Piacentini †

Abstract

Train movements in railway lines are generally controlled by human dispatchers. As disruptions often occur, dispatchers take real-time scheduling and routing decisions in the attempt to minimize deviations from the official timetable. This optimization problem is called Train Dispatching. We represent it as a Mixed Integer Linear Programming model, and solve it by a Benders’-like decomposition within a suitable master/slave scheme. Interestingly, the master and the slave problems correspond to a macroscopic and microscopic representation of the railway, recently exploited in heuristic approaches to the problem. The decomposition, along with some new modeling ideas, allowed us to solve real-life instances of practical interest to optimality in short computing time. Automatic dispatching systems based on our macro/micro decomposition - in which both master and slave are solved heuristically - have been in operation in several Italian lines since year 2011. The exact approach described in this paper outperforms such systems on our test-bed of real-life instances. Furthermore, a system based on another version of the exact decomposition approach has been in operation since February 2014 on a line in Norway.

1 Introduction

Although rail networks are complex and varied systems, in a simplified picture they may be viewed as a set of stations connected by tracks. Each train follows a specific route in this network, namely an alternating sequence of stations and tracks. Each railway operator develops a production plan for its trains, which is an expression of the railway operator’s desires and goals. Then, a following stage of interaction with the network operator results in an official timetable. In principle such timetable is conflict-free, that is, ensures no trains occupy simultaneously the same railway resource or different but incompatible resources. However, once put in operation, unpredictable events may occur, leading trains to deviate from the original plan. Furthermore, in

* SINTEF ICT, Oslo, e-mail: leonardo.lamorgese@sintef.no, carlo.mannino@sintef.no
† OptRail, Rome, Italy, e-mail: mauro.piacentini@optrail.com
such an interconnected system, deviations tend to propagate. To reduce the impact of such disturbances, re-routing and re-scheduling decisions must be taken, often in just a few seconds. This task is performed by specialized personnel, the dispatchers, who try to minimize (a measure of) the deviations from the original plan. Thus, dispatchers need to solve an optimization problem, and a very hard one at that. We refer to it as the Train Dispatching (TD) problem ([36]). Despite the relevance and (proven) computational complexity of such problem, human dispatchers are, almost everywhere, still not supported in their decision process. This may be rooted in the past inability of optimization systems to obtain sufficiently good solutions to present to dispatchers within the stringent time limit (or indeed at all), inevitably leading to a lack of "faith" in the concrete application of Operations Research to this domain. However, we show in this article that the TD problem can in fact be effectively tackled by suitable optimization techniques, leading to an improvement of the current status-quo, both in terms of "reaction" time and instances solved to optimality.

In short, the TD problem amounts to establishing, in real-time, a route and a schedule for each controlled train such that no conflicts occur with other trains and some function of the deviation from the official timetable is minimized. As such, the TD problem falls into the class of job-shop scheduling problems, which can be represented by suitable disjunctive formulations ([2]). In turn, such formulations can be casted into mixed integer linear programs (MILPs) by associating a binary variable with every pair of (potentially) conflicting operations and, for any such variable, a pair of alternative precedence constraints representing different orderings amongst operations.

The literature on TD is quite large and intermixed with that of the closely related problem Train Timetabling (TT), and it is out of the scope of this paper to give a detailed description. We refer the reader to recent surveys ([10]) or Ph.D. thesis’ ([21, 27]) for general discussions.

Every model for the TD problem must face the need of representing trains running through railway resources. Each such run is subdivided into atomic movements, which in turn are associated with the nodes of a path (route), as e.g. in [3] and [31]; since movements occur over time, atomic movements are also regarded as events (e.g. in [38, 40]). A schedule associates a specific time with each atomic movement, normally representing the start of the movement. A schedule must ensure that no trains occupy simultaneously non-sharable resources, such as a given platform in a station or a same track. To this end, any potential conflict in the use of a resource calls for a decision on "who goes first" ([38]). Such decisions give raise to disjunctive constraints and represent a major source of difficulty in solving TD problems. Disjunctive optimization problems can be modeled as MILPs and two alternative classes of formulations have been extensively exploited in the literature: time indexed formulations [19] and Big-M formulations. In time indexed formulations a binary variable is associated with each atomic movement and each of the time periods of a discretized dispatching horizon. Time indexed formulations (and time-expanded graphs) for train scheduling are adopted, e.g., in [4, 5, 7, 8, 6, 22, 28, 36, 43], and others. With the exception of [28],
all of these works deal with the train timetabling problem, which is solved off-line and where the number of time periods associated with train routes is reasonably small. However, in the (real-time) TD problem, the number of time periods associated with an atomic event may grow too large to be handled effectively by time-indexed formulations within the stringent times imposed by the application (for a discussion see [30]). Also the choice of the discretization step may be quite complicated and easily lead to solutions which are practically unattainable (see [22]).

In Big-M formulations, continuous variables are associated with the starting times of operations, whereas potential conflicts are represented by disjunctive precedence constraints which are casted into MILP models by associating a binary indicator variable with every disjunction and a pair of Big-M precedence constraints. A notorious flaw of Big-M formulations is their weakness, in the sense that they tend to return poor bounds, which in turn may result in very large search trees.

A second difficulty in solving the TD problem stems from the large number of potential routes for each train. The large dimension of time indexed formulations, the poor bounds of the Big-M formulations, and the large number of potential train routes, make exact solutions, especially by means of MILP based approaches, seldom sought and achieved in practice or in the literature. The few exceptions typically tackle small geographical regions (e.g. [15, 28, 30]) and a limited number of trains, even though they can contain complicated stations and junctions. However, the vast majority of works resort to heuristic techniques.

Recently, the macroscopic/microscopic principle (see [39]) has gained increasing attention and has been exploited in a number of works to tackle such huge difficult instances, both in dispatching and in timetabling. Roughly, it considers two aggregation levels of the original network. At a microscopic level, the railway infrastructure is given with a high degree of precision; all non-sharable resources are represented and the atomic movements of the trains are associated with every such resource on its route. At a macroscopic level, large portions of the network, typically stations, are represented as aggregated elements; for each train, the only interesting events are arrival time and departure time at and from each station, namely its timetable. The macro/micro approach can be sketched as follows: find a tentative, good timetable, then check if this is feasible at a microscopic level - namely there exists a microscopic solution matching the given timetable; if not, modify the original timetable and iterate. A few recent papers for the TD and TT problem fit into this framework. They differ in the way the macro and micro problems are modelled and solved, and in the way communication between micro and macro level is carried out. In [16], the macro problem is solved first by means of a mixed integer programming model (see [18] for details); the micro problem is attacked via heuristics capable to model in detail the movements of the trains across the station. If the micro solution matches with the macro solution, then we are done; otherwise some input parameters to the macro problem are modified according to the micro solution and the method iterates. In the approach presented in [17], [25] and [29], both the macro and the micro problems are formulated as mixed integer programming.
models (MILPs). However, while in [17] the solution (in sequence) of the two problems is embedded into a heuristic iterative mechanism, in [25], [29] and in this paper we develop an exact approach.

The new exact approach is based on the classical Benders’ decomposition principle or, better, its integer version as presented in [42] equipped with combinatorial Benders’ cuts introduced by Codato and Fischetti [11]. In our decomposition, the macroscopic problem (here denoted as Line Dispatching problem) acts as the master problem, whereas the microscopic problem (the Station Dispatching problem) is the slave. The Line Dispatching problem is defined on a simplified network, in which each station is replaced by a node, and is solved to optimality. The solution to the Line Dispatching problem produces, for each train, tentative arrival and departure times in the stations of the railway. The slave problem is a feasibility problem and amounts to finding, for each train and each station, a route and a schedule which are compatible with the tentative arrival and departure times; also, the schedule must ensure that no conflicts arise in the use of station resources. Similarly to [11], if the slave problem is infeasible, then a violated (combinatorial) cut in the variables of the master problem is identified and added to the master, and the process iterates. One fundamental property of the slave is that it naturally decomposes into many independent problems, one for each station. Each slave subproblem is then rather small and can be easily attacked.

Equipped with suitable delayed row and column generation schemes, this approach proved to be decisive for successfully tackling the real-life instances arising from a number of railway lines in Italy and Norway. The major advantages of the decomposition are shown to be the drastic reduction of the number of variables and constraints with respect to the standard, natural formulation and the freedom in modelling the Station Dispatching problem with suitable complexity, as the (general) Station Dispatching problem is NP-complete ([24, 25]). Also, since the slave problem is a feasibility problem, in most cases a solution can be found quickly by applying heuristic algorithms. Indeed, a semi-automated route setting system based on such decomposition, but with the exact integer programs replaced by local heuristics, has been in operation since 2011 on several Italian railway lines (see Section 6).

In this paper we also introduce a new and more effective way to model some conflicts in the Line Dispatching problem. In [25, 29], we associate decision variables with pairs of trains and every point of the line where such pair can possibly meet or pass\(^1\). Here, rather than establishing the exact point in which trains should meet or pass, the variables are associated with entire sub-regions. Through an iterative row generation mechanism, variables become active only if needed, i.e. if at the solution at the current iteration requires trains to share incompatible resources. Also, valid sets of strong linear inequalities are derived, improving the formulation and allowing tighter bounds to be found. Our computational results with the new approach improve the results presented in [25, 29]. Indeed we were able to solve to optimality previously unsolved hard instances and reducing computation times while, in all cases, respecting the time

\(^1\)In single-track lines these points can only be stations or sidings.
limit acceptable for dispatchers.

Now, we first summarize our overall contribution to the state-of-the-art, then detail the novel ideas introduced in this paper.

**New modelling perspectives.** The literature on train dispatching presents several MILP models. All must face the problem of representing atomic movements and time variables. Also, they need to represent the disjunctive decision "Who goes first in block \( b^n \) for a pair of conflicting trains \( i, j \). The decision corresponds to a (disjunctive) pair of precedence constraints on the schedule variables, exactly one of which will be satisfied by the final schedule. As already mentioned, the large majority of models for train dispatching (including our own) resort to continuous time variables to represent schedules and big-M constraints to represent disjunctions. Time-indexed formulations are in general preferred for timetabling, with few exceptions, as for example in [28] where a binary variable is associated with a specific time period and an atomic movement. An innovative, time-indexed representation was introduced by a group active at ETH university in Zurich (see, e.g., [8, 9, 21]), where binary, time-indexed variables are associated to a sequence of movements rather than to atomic ones.

Typically, when continuous variables are chosen for atomic movements (as, e.g., in [20, 32, 34, 40]), a binary variable \( x_{ij}^b \) is associated with each disjunction; depending on its value, exactly one of the two constraints will be active (and the other one redundant). In contrast, our binary variables are associated with a different decision, namely "where" a pair of potentially conflicting trains will have to meet on the line. Also, rather than having the standard (binary) disjunction between two precedence constraints, we have a multiple disjunction among sets of precedence constraints. Each binary variable is then associated to a set in a multi-disjunction: if the variable takes value 1, then all the constraints in the set become active (otherwise they are all redundant). This change in perspective w.r.t. the "who goes first" approach allowed us for a drastic reduction in the number of binary variables and, most important, for an efficient implementation of row generation techniques.

**New algorithmic enhancements.** Most papers presented in the dispatching literature focus on modelling issues. Only a few of them, basically devoted to heuristics, focus on developing new solution approaches (among these the recent papers [12, 13, 14, 41] and the already mentioned work by the ETH group [8, 9, 21]). Even if our modelling innovations are quite important, still we believe our major contributions lie in the development of effective algorithms. First, we reduce the macroscopic/microscopic approach to a very well known decomposition scheme, namely Benders’ reformulation. This allows us to define a suitable, exact master/slave approach. In addition, also thanks to our modeling innovations, we can effectively apply delayed row and column generation in the sub-problems.
**Practice.** Even though the literature on train dispatching is already very large, there is a general lack of real-life implementations, as also remarked in two recent studies ([10, 34]). Therefore, our applications represent a breakthrough in the current practice. Dispatchers are in fact generally required to take decisions "manually" or with the support of simple IT tools, following rules of thumb or heuristic principles. Exception to this practice is a tool based on our exact decomposition approach, put in operation in Norway in February 2014, to support dispatching on the Stavanger-Moi line (described in [25]). Furthermore, a system based on a heuristic version of this approach has been in operation in Italy since 2011 ([29]).

The modelling ideas and decomposition approach presented in this paper were partly introduced in earlier works [29, 25, 26]. However, the specific contributions of this paper are manyfold:

- **Modelling.** We introduce a new way to model the meet event for trains running in opposite directions (a draft version was presented in ([26])). We also introduce a better way to represent relations between trains running in the same direction. Furthermore, we model more complex station configurations by an ad-hoc MIP formulation (derived from the equivalence with list coloring on interval graphs introduced in [25]).

- **Algorithmic.** We introduce a new heuristic to solve the separation problem for stations (slave problem). We also introduce several enhancements to the master-slave algorithm and a way to exploit previous solutions in a dynamic context, where the algorithm is invoked several times in a minute and previous solutions are at hand.

- **Systematization.** Many works in the literature refer to the macroscopic/microscopic approach. In this paper we clarify the relation between this algorithmic decomposition and the classical reformulation proposed by Benders.

- **Practice.** We show that the novelties introduced in this paper allow to further speed up the solution process, crucial aspect in a real-time application.

## 2 Problem description

A railway network is generally a complex, interconnected system with different types of infrastructure which can be represented with growing detail. Moving from the macroscopic to the microscopic level inevitably increases the number and complexity of resources to be taken in account, turning even simple representation into a non trivial task. Although many of such details are relevant in a practical implementation, they are irrelevant to the understanding of our approach and may be omitted while describing our models.
In this simplified picture, we start by modeling the Railway Network as a set $S$ of stations and a set $C$ of connections between pairs of stations. In the general case, adjacent stations may be connected by an arbitrary number of parallel tracks. In this paper however, we will consider at most two tracks (double-track). Let us introduce next the elements of a railway network on the macroscopic and microscopic infrastructural level.

**Stations.** A station corresponds to a region of the railway network where trains can stop to perform tasks as embark and alight passengers, meet or pass other trains, do maintenance operations, etc. To our purpose, a station can be represented by a directed graph, with nodes associated with tracks where trains can stop to execute some operations (stopping points) and directed edges associated with interlocking-routes, i.e. tracks connecting stopping points. Special stopping points are the entry/exit points, where trains enter or exit the station, and the platforms, where trains may embark and alight passengers, or simply wait before departure. Each train running through a station $s$ is associated with a directed path in the station graph. In the common case of passenger trains, such path contains a platform node. Trains running through a station $s$ without a scheduled stop (ex. freight trains) may or may not have platforms in their path. Also, a path through a station will always contain an entrance (exit) node, unless the train originates (or terminates) in the station.

**Connections.** A track is a rail used by rolling stock vehicles. One or more tracks which connect two stations of the line are a connection. Tracks may be partitioned into smaller units called sections, which, for safety reasons, can be occupied by a single train at the time. Generally, trains running on a track in the same direction must be separated by a minimum number of sections. Hereafter we only consider single section$^2$ tracks. Also, we assume that rolling stock running on a double-track connection may use each track exclusively in one direction (uni-directional traffic), unless in the case of failure of one of the two tracks (interruption). Should this occur, trains running in opposite direction will alternate on the available track, as for the case of single-track connections. Although the number of railways which allow the use of both directions for each track (bi-directional traffic) is growing, this assumption still holds in the majority of cases and in all instances of our real-life test-set.

**Trains.** Let $T$ be the set of controlled trains. By controlled trains we intend trains for which, at time zero (i.e. the observed time) information is available and re-scheduling and re-routing decisions can be taken. Note that many of such trains may still have to enter the controlled line. Moreover, TD is a real-time activity, which means that a controlled train can be anywhere in the railway when the solution process is started, or can be simply expected to originate somewhere at a given time in the future.

$^2$the extension to the general case is trivial
Train Routes. A train route corresponds to the sequence of tracks, interlocking routes and stopping points encountered by a train in its run through the railway. At a macroscopic level, the route of a train $i \in T$ is an alternating (ordered) sequence of stations and connections. At a given time, we denote by $S(i)$ the set of stations on the route of $i$. There is a natural ordering $(S(i), \succ_i)$, where for any distinct $r, s \in S(i)$ we let $s \succ_i r$ if $s$ is encountered after $r$ on the route of train $i$. We assume the set is simple, i.e. any station can be encountered at most once. More complicated routes, such as circular routes, can always be decomposed as a concatenation of simple routes, and they are quite rare in railway timetables. We then define the set $C(i)$ of connections in the route of train $i$, i.e. $C(i)$ contains a connection for each pair of consecutive stations in $S(i)$, plus possibly the initial connection leading to the first station (if the current position of $i$ is on a track). Also the connections in $C(i)$ are naturally ordered and for $c, d \in C(i)$ we write $d \succ_i c$ if $d$ is encountered after $c$ on the route of train $i$. Also, if the train is currently outside the controlled region, we add to $C(i)$ a fictitious connection to represent it, which is the first one in the ordering of $C(i)$. Similarly, if the train will end outside the controlled region we add a second fictitious connection which will be the last one in the ordering of $C(i)$.

Let us consider now two distinct trains $i$ and $j$ and let $S(i, j) = S(i) \cap S(j)$ be the set of stations common to the routes of $i$ and $j$. We assume that $S(i, j)$ is a set of consecutive stations both for $i$ and for $j$, and we order the stations in $S(i, j)$ as they are ordered in $S(i)$. Trains $i$ and $j$ are called followers if train $i$ visits the stations in $S(i, j)$ in the same order as train $j$; otherwise they are called opposite trains. We denote by $Opp$ the set of unordered pairs of opposite trains and by $Foll$ the set of unordered pairs of follower trains. Let $C(i, j)$ be the set of connections between consecutive stations in $S(i, j)$. Also the connections in $C(i, j)$ are ordered as encountered by train $i$. Each connection can be single or double-track, and we denote by $DC(i, j)$ the set of double tracks between $C(i, j)$, ordered in the usual way.

At microscopic level, routes also include the complete trajectories of trains across the stations, defined as alternating sequences of nodes and arcs of the station graph.

Official and real-time timetable. The schedule is a real vector $t$ which specifies the exact timing of all the movements of the train’s route. A given component $t_k$ of the schedule $t$ is associated with a train $i$ and with an non-sharable railway resource $r$, and denotes the time at which $i$ enters $r$. Non-sharable railway resources include track sections, station platforms or stopping points, converging interlocking routes, etc. A resource is occupied by a train until it enters the next resource on its route. A timetable is the sub-vector of a schedule specifying when trains arrive and depart from each station in their route. The official timetable is the wanted one, typically planned long in advance. The real-time timetable, instead, is the outcome of real-time dispatching decisions.
Conflicts and Meet and Pass events. As mentioned above, double-track connections generally only allow uni-directional traffic on each track. Even though, in principle, trains may run both tracks in the same directions and so followers may pass or meet on double track connections, this is, with few exceptions, an undesirable operation and railway operators only allow human dispatchers to perform it. So, we assume here that only opposite trains may cross each other on double track connections. Follow trains can meet and pass each other in stations several times. When running on tracks, followers must maintain a safety distance of at least one section.

Conflicts are associated with schedules. We say that a schedule $t$ generates a conflict if, according to $t$, two trains occupy simultaneously a non-sharable railway resource. So, in order to be implemented, a schedule must be conflict-free.

Dispatching horizon. The dispatching horizon is the time window which is taken in account for decision making. The real-time nature of the problem implies that the origin of the time window is the instant in time at which the observation is made. There are several, conflicting factors which have an impact on defining its duration. Since, in principle, each re-scheduling or re-routing decision may have an impact on future events, considering a wider time window can present advantages. On the other hand, the need for fast computation and the growing uncertainty related to events lying in the future suggest that choosing a narrower time window could be more appropriate. In other words, choosing the dispatching horizon is a delicate and open topic which, in any case, must be thoroughly discussed and agreed with railway operators.

Objective Function Conformity to the official timetable emerged, from discussions with the Italian railway network operator (RFI) engineers, as the main factor in determining the quality of a real-time schedule $t$. Consequently, we agreed to define the cost function $c(t)$ as a sum of terms associated with trains, where each term is convex and piece-wise linear with the delay of the associated train. Furthermore, each train may be assigned a specific weight in the objective, based on its relative priority. We remark here that alternative objectives can be easily taken into account. For example, one may want to minimize the number of (heavily) delayed trains or the cost of lost connections (as in [38]).

The Train Dispatching problem We are now able to state the TD problem:

**Problem 2.1** Given a railway infrastructure and its current status, a set of trains and their current position, find a route for every train from its current position to the destination, and find an associated real-time timetable so that the cost function $c(t)$ is minimized.

We remark here that different versions of the TD are presented in the literature. One of the most relevant is probably the so called Delay Management: in its basic version,
TD is extended with decisions on which connections (between feeder and receiver trains) should be maintained in case of delays, so as to minimize the inconvenience for the passengers (see [38]). The basic version has been then extended in a sequence of papers to take into account station capacities [17] and the fact that passengers may choose different routes [18]. Finally, a first step toward the full integration of TD and delay management is discussed in [16].

In order to solve the TD problem, we need to solve both a routing and a scheduling problem. In our model and real-life instances, the routing problem is restricted to stations, as connections are either single-track or double-track with track fixed for each direction. The TD problem can be easily modeled by Mixed Integer Linear Programming (MILP) formulations (as in [30] or [17]). However, TD instances of practical interest are typically too large to be attacked directly by state-of-art commercial solvers, and most authors resort to heuristic approaches or to simplified versions of the problem.

We have followed a different path, developing a decomposition technique which makes it possible to apply classical MILP techniques and solve to optimality instances of the TD problem of practical interest. Here we further develop the approach we introduced in [25, 26, 29], namely we identify two major sub-problems: The real-time Line Dispatching problem, which amounts to establishing a schedule for the trains so they only meet in stations or in double track regions (or they do not meet at all), minimizing a given cost function; And the real-time Station Dispatching problem, a feasibility problem which amounts to finding suitable routes in the station and a schedule which matches the given timetable. In the next sections we will show how the TD problem can be formulated as a MILP where the 0,1 variables (say $x$) are associated with decisions of type "where train $a$ and train $b$ shall meet" or "which platform is to be assigned to train $i$ in station $s$". The schedule is represented by a real vector $t$. Now, the MILP associated with a TD problem typically exhibits a block structure as follows:

$$
\begin{align}
\min & \quad c(t) \\
\text{s.t.} & \quad Ax^L + Bt \geq b, \\
& \quad Dt + Ex^S \geq q, \\
& \quad t \text{ real, } x^L, x^S \text{ binary}
\end{align}
$$

where the first block (i) (plus (iii)) is associated with the Line Dispatching problem and the second block (ii) (and (iii)) is associated with Station Dispatching problem. Remarkably, the two blocks "communicate" through a small subset of the continuous variables $t$, namely those associated with the timetable. In other words, fixing trains arrival and departure times at and from the stations, perfectly decomposes the original MILP into (many) smaller programs. As mentioned in the introduction, our methodology can be described as an adaptation of the classical Benders' reformulation to cope with integer slave problems - as discussed in [42]. First, block (ii) is neglected, the problem restricted to (i) and (iii) is solved to optimality, and a schedule $t^*$ is found. If
$t^*$ can be extended to a solution satisfying (ii) and (iii), then we are done. Otherwise a suitable feasibility cut - a combinatorial analogue to the classical feasibility Benders’ cut in the Benders’ decomposition approach (see [33]) - is generated, added to the restricted problem and the process is iterated. Combinatorial Benders’ cuts for the general case are introduced and discussed in [11]; in this paper, however, the slave problem is a linear program and the combinatorial cut may be viewed as a strengthening of the standard Benders’ feasibility cut.

In the next sections we describe in detail the two problems in this decomposition and how combinatorial feasibility cuts are generated.

3 The Line Dispatching problem

First we discuss the Line Dispatching problem. In this problem, every station is modeled as a black box and the precise movements of trains within each station are ignored, except for the (minimum) time necessary to the train to cross the station. What matters at this stage is only when trains enter and exit the station, namely the real-time timetable. So, for each train $i \in T$ and each station $s \in S(i)$, we introduce two continuous variables $a^i_s$ and $d^i_s$, representing, respectively, the arrival and departure time of $i$ at and from $s$. The vector $(a, d) \in \mathbb{R}^{2|T}$ is the real-time timetable. Next, we introduce the major constraints.

**Single Train Simple Precedence Constraints.** Denoting by $W^i_s$ the minimum time necessary for $i$ to cross station $s$ (including the time needed to embark and alight passengers)

\[
d^i_s - a^i_s \geq W^i_s
\]

Also, denoting by $Q^i_k$ the minimum running time for train $i$ to run the track from station $s_k$ to station $s_{k+1}$, we have:

\[
a^i_{s_{k+1}} - d^i_{s_k} \geq Q^i_k
\]

**Modeling meet and pass events.** First we proceed as in [25]. Namely, for any pair of trains $\{i, j\} \in \cup \text{Foll}$ and every station $s \in S(i, j)$ we introduce a binary variable $y^i_j$ which is 1 if and only if $i$ and $j$ meet in $s$. Meeting in $s \in S(i, j)$ implies that $i$ enters $s$ before $j$ leaves $s$ and vice versa. This can be expressed by the following pair of constraints:

\[
\begin{align*}
(i) \quad d^i_s - a^i_s & \geq (y^i_j - 1)M \quad \{i, j\} \in \text{Opp} \cup \text{Foll}, s \in S(i, j) \\
(ii) \quad d^j_s - a^j_s & \geq (y^i_j - 1)M \quad \{i, j\} \in \text{Opp} \cup \text{Foll}, s \in S(i, j)
\end{align*}
\]

To simplify the description we assume here that the arrival time is the time the train enters the station, and the departure time is the time the train leaves the station.
where $M$ is a suitably large constant so that the constraints become redundant when $y_{s}^{ij} = 0$.

**Meet vectors and meet graphs.** The vector $y$ plays a crucial role in our decomposition approach. The subvector $y_{s}$ corresponding to the components of $y$ associated with a given station $s$ is called station $s$ meet vector. We associate with $y_{s}$ an undirected graph $G(y_{s})$, the station $s$ meet graph, with vertex set $V = T$ and edge set $E(y_{s}) = \{(i, j) \subseteq V : y_{s}^{ij} = 1\}$, i.e. there is an edge connecting node $i$ and node $j$ in $G(y_{s})$ if and only if the corresponding trains meet in $s$ according to $y_{s}$. It is not difficult to see ([29]) that $G(y_{s})$ is an interval graph and $y_{s}$ is then the incidence vector of the edges of an interval graph.

**Opposite Trains.** Opposite trains can also cross in double-track connections. For each pair of opposite trains $\{i, j\} \in Opp$ and every double track $d \in DT(i, j)$ we introduce a binary variable $z_{d}^{ij}$ which is 1 if and only if $i$ and $j$ cross in $d$. Denoting by $f(d)$ and $s(d)$ the first and the second station (according to the ordering of $S(i, j)$) at the ends of the double track connection $d$, we have a pair of constraints similar to its station counterpart (4) but "centered" on track $d$:

\[
\begin{align*}
(i) & \quad a_{s(d)}^{i} - d_{s(d)}^{i} \geq (z_{d}^{ij} - 1)M & \{i, j\} \in Opp, d \in DC(i, j) \\
(ii) & \quad a_{f(d)}^{j} - d_{f(d)}^{j} \geq (z_{d}^{ij} - 1)M & \{i, j\} \in Opp, d \in DC(i, j)
\end{align*}
\]  

The above pair of constraints ensure that when $i$ and $j$ meet in $d$, $i$ enters $d$ before $j$ leaves it and viceversa.

Note that since $S(i, j)$ also contains the external, uncontrolled regions, any pair of opposite trains must actually meet somewhere, and we have

\[
\sum_{s \in S(i,j)} y_{s}^{ij} + \sum_{d \in DC(i,j)} z_{d}^{ij} = 1 \quad \{i, j\} \in Opp
\]  

Next, we have to ensure that if trains $i$ and $j$ do not meet in station $s$, then the leading train exits $s$ before the trailing one enters $s$. It is not difficult to see that this condition is always satisfied by opposite trains for any real-time timetable satisfying (5) and (6). The situation is different for followers.

**Followers.** For any $\{i, j\} \in Foll$, the above condition must be enforced somehow on the real-time timetable. First, we assume w.l.o.g. that $i$ is the leading train when the trains first appear in the common region. Then, for each station $s \in S(i, j)$ we introduce a binary variable $o_{s}^{ij}$ with $o_{s}^{ij} = 1$ if $i$ enters $s$ before $j$ and $o_{s}^{ij} = 0$ if $j$ enters $s$ before $i$. Now, let stations $s, q \in S(i, j)$ with $q$ next to $s$ in the routes of $i$ and $j$. Then the relative order when entering $q$ can differ from the relative order when entering $s$ only if the trains meet in $s$. This fact can be easily expressed by the following pair of constraints:
\[(i) \quad o^i_j \leq o^i_s + y^i_j \quad \{i, j\} \in \text{Foll}, \quad s \in \tilde{S}(i, j), \quad q = \text{succ}_i(s)\]
\[(ii) \quad o^j_i \leq o^j_q + y^j_i \quad \{i, j\} \in \text{Foll}, \quad s \in \tilde{S}(i, j), \quad q = \text{succ}_i(s)\]

where $\tilde{S}(i, j)$ is $S(i, j)$ without its last station, and $\text{succ}_i(s)$ is the station immediately after $s$ according to $\succ_i$.

Now if $i$ and $j$ do not meet in $s$ ($y^i_j = 0$) and $i$ is leading in $s$ ($o^i_s = 1$), then $i$ exits $s$ before $j$ enters $s$:

\[a^i_s - d^i_s \geq (o^i_s - y^i_s - 1)M \quad \{i, j\} \in \text{Foll}, s \in S(i, j)\] (8)

In all other cases the above constraint becomes redundant.

Similarly, if $i$ and $j$ do not meet in $s$ and $j$ is leading in $s$, then $j$ exits $s$ before $i$ enters $s$:

\[a^j_s - d^j_s \geq (-o^j_s - y^j_s)M \quad \{i, j\} \in \text{Foll}, s \in S(i, j)\] (9)

Finally, followers’ timetable must respect headway safety rules. This requires the leading train always to be at least one section ahead of the follower. So, let $d \in C(i, j)$ be a connection common to followers $i$ and $j$ and let as before $f(d)$ and $s(d)$ denote the first and the second station at the extremes of connection $d$, respectively. Since we only consider single section tracks, the leading train must leave $d$, so entering station $s(d)$, before the follower enters $d$, so exiting $f(d)$. Note that the leading train in station $s(d)$ is also leading in track $d$. Then we have:

\[d^j_{f(d)} - a^i_{s(d)} \geq (o^j_{s(d)} - 1)M \quad \{i, j\} \in \text{Foll}, d \in C(i, j)\] (10)

and:

\[d^i_{f(d)} - a^j_{s(d)} \geq -M \cdot o^i_{s(d)} \quad \{i, j\} \in \text{Foll}, d \in C(i, j)\] (11)

### 3.1 Enhancing the model

The model so far introduced contains in general a large number of binary variables and, even worse, Big M constraints. As mentioned in the introduction and discussed in detail in a next section, we tackle this problem by delayed variable and constraint generation. The binary variables are needed to model decisions in order to avoid conflicts between pairs of trains and to pick among precedence constraints in disjunctions. Now, simple tests immediately reveal that most difficulties in the solution process are caused by pairs of opposite trains. In fact, a slight delay from the official plan for a given train is not likely to produce effects on many other trains running in the same direction. Indeed the number of take-overs is typically very small. On the other hand, a small deviation may result in many potential conflicts with trains running in the opposite direction. Next, we introduce a new modeling idea to tackle conflicts for opposite trains which effectively contributes to reducing the computational burden. Indeed, statistics
related to conflicting trains show that the average number of conflicting opposite pairs is significantly higher than the average number of conflicting follower pairs. Namely, for one of the Italian lines considered in this work (see Section 6), the ratio is 6:1 whereas for the other it is 5:2.

If we consider a pair of opposite trains \( \{i, j\} \) and a connection \( c \in C(i, j) \), then \( i \) and \( j \) may either cross (i) before \( c \), i.e. in a station or in a double track encountered by \( i \) before \( c \), or (ii) after \( c \). If \( c \) is a double-track connection, then case (ii) also includes the possibility that \( i \) and \( j \) meet in \( c \). In order to model these two possible options, we introduce a binary variable \( w_{ij}^c \) for each pair of opposite trains \( \{i, j\} \in \text{Opp} \) and each connection \( c \in C(i, j) \), with \( w_{ij}^c = 1 \) if \( i \) and \( j \) meet before \( c \) and \( w_{ij}^c = 0 \) otherwise (\( i \) and \( j \) meet after \( c \) or in \( c \)).

Denoting by \( f(c) \) the station which immediately precedes (follows) connection \( c \) in the usual ordering, then we have:

- Case \( w_{ij}^c = 1 \) (\( i \) and \( j \) meet before track \( c \)) implies that train \( j \) enters \( f(c) \) before \( i \) exits \( f(c) \), which is modeled by the following constraint:

\[
d_{f(c)}^i \geq a_{f(c)}^j + (w_{ij}^c - 1)M \quad \{i, j\} \in \text{Opp}, \quad c \in C(i, j)
\]  

(12)

- Case \( w_{ij}^c = 0 \) (\( i \) and \( j \) meet in track \( c \) or after). If \( c \) is not double-track then \( i \) and \( j \) meet after \( c \), and train \( i \) enters \( s(c) \) before \( j \) exits \( s(c) \). This can be modeled by the following \( \text{Big-M} \) constraint:

\[
d_{s(c)}^i \geq a_{s(c)}^j - w_{ij}^c M \quad \{i, j\} \in \text{Opp}, \quad c \in C(i, j) \setminus DC(i, j)
\]  

(13)

If \( c \) is double-track, then \( i \) and \( j \) can also meet in \( c \) and the above constraint must be amended as follows:

\[
d_{s(c)}^i \geq a_{s(c)}^j - (w_{ij}^c + z_{ij}^c) M \quad \{i, j\} \in \text{Opp}, \quad c \in DC(i, j)
\]  

(14)

In fact, if \( i \) and \( j \) meet in \( c \) (\( z_{ij}^c = 1 \)), then \( d_{s(c)}^i < a_{s(c)}^j \) holds, and (14) becomes redundant. In other words, \( d_{s(c)}^i \geq a_{s(c)}^j \) holds only if \( i \) and \( j \) do not meet "before" \( c \) (i.e. \( w_{ij}^c = 1 \)) or in \( c \) (i.e. \( z_{ij}^c = 1 \)).

Adding these new variables and constraints to the model allows us to neglect the \( y \) variables and drop the associated constraints (4) and (6). Also, we can easily strengthen the model by observing that, if \( c, d \in C(i, j) \) and track \( d \) is after \( c \), then, if two opposite trains \( i \) and \( j \) meet before \( c \), they meet before \( d \) as well. So we have

\[
w_{d}^{ij} \geq w_{c}^{ij}, \quad \{i, j\} \in \text{Opp}, \quad c, d \in C(i, j), \quad d \succ_i c
\]  

(15)

A final observation concerns the meet vector \( y \), which plays a crucial role in the Station Dispatching problem to be discussed in the next section. Even if the \( y \) variables
are neglected for opposite trains, they can be easily derived from the $w$ and the $z$ variables.

4  Modeling the Station Dispatching problem

In our decomposition scheme, a solution to the Line Dispatching problem provides a timetable for the controlled trains, namely the arrival and departure time of every train at and from each station of its route. We now need to verify if such timetable is actually achievable, that is, for any station $s \in S$ and every train $i \in T$ through $s$, we need to find a routing and a scheduling of all of the movements of $i$ through $s$ so that all arrival and departure times match with the timetable and no conflicts in the use of resources arise. This problem closely resembles its off-line version, the *Train Platforming* problem, see [6]. Actually, in most small/medium sized stations, there is only one path from the entry point to a given platform and from the given platform to the exit point. So, there is a unique route from the entry point to the exit point through a given platform. This is the case of the instances considered in Section 6.

Another simplifying assumption that we introduce is the following:

**Assumption 4.1** For any train $i \in T$, the minimum time $W_i^s$ needed by the train to traverse station $s$ does not depend on its route.

Assumption (4.1) is needed to simplify the interaction between the slave and the master in our decomposition and is typically satisfied in most small/medium sized stations. Even when the assumption is not fulfilled, typically in large stations, the decomposition approach provides us a lower bound on the optimal solution value; in addition, the solutions produced can normally be turned into feasible solutions, most often of good quality. The above assumption is violated, for instance, when the length of the interlocking routes from the entry point to the platforms significantly differ from platform to platform, or when a train stops before reaching the platform due to potential conflicts with other trains in the station.

When Assumption 4.1 is satisfied, the Station Dispatching problem is reduced to deciding whether the platforms in a station suffice to accommodate all incoming trains. In particular, if two distinct trains $i, j$ happen to traverse or stop in the station at the same time, they must be assigned different platforms.

We state now the Station Dispatching problem for a station $s \in S$. Remark that the solution to the Line Dispatching problem provides us, besides a timetable, also the associated meet vector $y_s$, which is the incidence vector of the train pairs meeting in $s$, and the corresponding meet graph $G(y_s) = (T, E(y_s))$. Actually, the timetable for $s$ immediately provides an interval representation of $G(y_s)$, where for each train the extremes of the associated interval are the arrival and departure time, respectively.

---

We must keep in mind that in practical applications small deviations may be neglected.

Again this assumption may be violated in large stations.
Problem 4.2 ([25]) Let $P$ be the set of platforms of station $s$, let $T$ be the set of controlled trains and let $y_s$ be a meet vector in station $s$. For every train $i \in T$ denote by $P(i) \subseteq P$ the set of platforms that can accommodate $i$. Assign to each $i \in T$ a platform in $P(i)$ so that $i$ and $j$ receive a different platform whenever $y_{ij} = 1$ (i.e. $\{i, j\} \in E(y_s)$), or prove that no such assignment exists.

It can easily be shown that the above problem is NP complete by reduction from list coloring of interval graphs (see [25]). The problem can be formulated as a MILP by introducing a binary variable $q_{ip}$ for each $i \in T$ and each $p \in P(i)$ which is 1 if and only if platform $p$ is assigned to train $i$. Then $q$ is a feasible assignment if and only if it satisfies:

\begin{align}
(i) \quad q_{ip} + q_{jp} &\leq 1, \quad \{i, j\} \in E(y_s), p \in P(i) \cap P(j) \\
(ii) \quad q_{ip} &\in \{0, 1\}, \quad i \in T, p \in P(i)
\end{align} 

(16)

However, the problem becomes easy ([29, 25]) when $P(i) = P$ for all trains $i$, i.e. every train can be accommodated in any of the platforms of the station (we call this case the all-good Station Dispatching problem). In fact, an assignment exists if and only if $G(y_s)$ can be colored with $|P|$ colors, which in turn can be done easily since $G(y_s)$ is interval. Namely, $G(y)$ can be colored with $|P|$ colors if and only if $G(y_s)$ does not contain a clique $K$ with $|K| > |P|$. Since $G(y_s)$ is interval, this test can be performed in polynomial time. When $P(i) \neq P$ holds for some $i \in T$, then the condition is not sufficient anymore, but stays necessary, that is a feasible assignment exists only if $G(y_s)$ does not contain a clique $K$ with $|K| > |P|$. We will use this necessary condition in our solution algorithm.

When Assumption 4.1 is not satisfied, or simply when trains simultaneously in station can be assigned the same platform (by letting the second train wait somewhere in the station), then a more complicated version of the Station Dispatching problem arises and we cannot limit ourselves to solving Problem 4.2. However, all our real-life instances refer to small-medium stations and satisfy the condition.

Finally, observe that if $G(y_s)$ is not connected, then we can decompose the original problem into independent sub-problems, each associated with a connected component. This situation often occurs in small stations, as it occurs if there are periods of the dispatching horizon when the station is empty of trains.

4.1 An algorithm for the Station Dispatching problem.

We first remark once more that the Station Dispatching problem is a feasibility problem, which means that any feasible assignment is a solution to the problem. Next, we summarize the solution algorithm:

1. Find all connected components in $G(y_s)$.

2. For each connected component $H$ do:
(a) Find a maximum cardinality clique $K$ in $H$. If $|K| > |P|$ return FALSE
(b) Find an assignment $q_H$ by applying a heuristic method.
(c) If $q_H$ not found
   Solve system (16) associated with $H$. If infeasible, return FALSE.

3. A feasible assignment $q$ is found. Return TRUE.

We discuss now in detail each step of the above algorithm.

**Find all connected components in $G(y_s)$.** Since $G(y_s)$ is interval and an interval representation is at hand, this problem can be solved in linear time in $|T|$.

**Find a maximum cardinality clique $K$ in $H$.** Since $H$ is an interval graph, also this step can be performed in linear time in $|T|$, see [37].

**A heuristic search for Station Dispatching.** The Station Dispatching problem is a list coloring problem, and thus any algorithm for list-coloring on (interval) graphs may be applied. In our current implementation, we simply resorted to a greedy algorithm. Namely, we visit trains by ascending cardinality of available platforms, assign to the current train the first available platform and adjust all other available platforms accordingly.

**Solve the 0,1 program associated with the problem.** To solve this problem we use IBM ILOG CPLEX (R) as commercial solver. The solver implements a branch&cut algorithm. Since we deal with a feasibility problem, the search can be halted as soon as a feasible solution is found. In any case, the real-life instances of our test-bed are easily tackled by CPLEX, as the number of platforms is relatively small.

**Combinatorial cuts.** Since the feasibility of program (16) only depends on trains meeting in $s$, if (16) is infeasible for some meet vector $\bar{y}_s$ then at least two of the trains meeting in $s$ (according to $\bar{y}_s$) cannot meet there. In other words, any feasible meet vector $y_s$ must satisfy the following linear constraint:

$$\sum_{\{i,j\} \in E} y_{ij}^s \leq |E| - 1$$

where $E$ is the set of edges of $G(\bar{y}_s)$.

In principle, the above constraint may contain a large number of variables, since many pairs of trains may meet in a station $s$ during the dispatching horizon. However, in many practical situations, the infeasibility is caused by a small set of trains. This is the case, for example, when the infeasibility arises because $G(\bar{y}_s)$ contains a clique $K$.  

17
with \(|K| > |P|\), which can be tested efficiently. In this case we may let \(E\) in (17) be the edges in clique \(K\) and we may rewrite (17) as:

\[
\sum_{\{i,j\} \subseteq K} y_{ij}^s \leq \left(\frac{|K|}{2}\right) - 1
\]  

(18)

In many cases, typically arising in small stations \(s \in S\) for regional or minor lines, the graph \(G(\bar{y}_s)\) is the union of several connected components, namely there are periods when the station is empty of trains. Then, if the problem is infeasible, for at least one connected component \(H\) a feasible assignment does not exist. In this case, we may restrict (17) to the edges in \(H\).

5 Solution Algorithm

We are finally able to describe the overall solution algorithm. As mentioned above, we apply the classical master-slave decomposition approach. The master role is played by the Line Dispatching problem, more precisely by the MILP problem defined by the inequalities from (2) to (15) and by the combinatorial cuts generated by the slave. In turn, the slave corresponds to the solution of \(|S|\) Station Dispatching problems, one for each station: if not feasible, the slave will return one or more new combinatorial cuts to be included into the master problem. The master-slave pair is solved in sequence several times, until the last slave problem will be feasible - in which case the current solution is the optimal one - or until the current master turns out to be infeasible, which implies that there exists no solution to the overall TD problem. At iteration \(k\) the master program also contains a family \(E^k\) of constraints of type (17): each constraint in \(E^k\) is associated with a set of train pairs \(E\). We denote\(^6\) by \(E^k = \{E^k_1, E^k_2, \ldots\}\) the family of sets associated with the combinatorial cuts at iteration \(k\) and we have \(\emptyset = E^0 \subseteq E^1 \subseteq \ldots\).

Solving the current master problem. A major difficulty arises in solution of the master problem. Indeed, the pairs of opposite (\(|Opp|\)) and follower trains (\(|Foll|\)) grow quadratically with the number of trains. Consequently, when dealing with a large number of trains, the number of Big\(_M\) constraints from (4) to (15) and the number of related variables \(y, w, z\) grows very large. The relaxations of the resulting MILPs tend to be very weak making them hard to solve in practice. To cope with this, we apply delayed row and column generation. Namely, we solve a sequence of restricted MILPs \(M^0, M^1, \ldots\) containing only subsets of the variables and constraints of the problem. In order to represent the programs in the sequence, observe that each of the constraints from (4) to (15) is completely identified either by a set of opposite train pairs \(Opp\), or by a set of follower pairs \(Foll\). On the other hand, constraints (17) are identified

\(^6\)With a slight abuse of notation, we let \(E^k\) denote both the family of constraints and the family of sets of train pairs in one-to-one correspondence with the constraints.
So, we denote a restricted master program by \( M(O, F, \mathcal{E}) \), where \( O \subseteq OPP \), \( F \subseteq \text{Foll} \), and \( \mathcal{E} \) is a family of sets of train pairs. The MILP \( M(O, F, \mathcal{E}) \) is defined by including all the inequalities (2) and (3) along with the corresponding variables, all combinatorial cuts associated with the sets in \( \mathcal{E} \) and the subset of inequalities (and variables) (4) to (15) associated with sets \( O \) and \( F \).

At each iteration of our algorithm we solve the current restricted master \( M = M(O, F, \mathcal{E}) \). If \( M \) is infeasible, then it is easy to see that the original TD problem has no solution. Otherwise, let \( t^* = (a^*, d^*) \) be the current optimal schedule. By inspection one can easily verify if \( t^* \) generates conflicts for some pairs of opposite trains \( \bar{O} \). Similarly, we can identify a subset \( \bar{F} \in \text{Foll} \) of follower pairs involved in conflicts generated by \( t^* \). If at least one of the two sets \( \bar{O}, \bar{F} \) is non-empty, we add \( \bar{O} \) to \( O \), \( \bar{F} \) to \( F \), construct the new restricted master problem associated with such sets (and the former \( \mathcal{E} \)) and iterate. Otherwise, \( t^* \) is conflict-free for the Line Dispatching problem and we need to check if it is possible to fulfill the associated timetable for every train and station by solving the slave problem. Namely, for each station \( s \in S \), a timetable\(^7\) \( t^* \) is extracted from \( t^* \) and \( |S| \) Station Dispatching problems are solved to test the feasibility of each individual \( t^* \). If one or more timetables are not feasible, the slave problem returns a set \( \bar{\mathcal{E}} \) of combinatorial cuts violated by the current solution. The set \( \bar{\mathcal{E}} \) is added to \( \mathcal{E} \), a new restricted master is generated and the method iterates. Next, we summarize our Master-Slave approach to the TD problem:

**Optimal Train Dispatching**

0. Set \( i = 0 \); Set \( O = \emptyset \), \( F = \emptyset \), \( \mathcal{E} = \emptyset \)

1. Solve \( M(O, F, \mathcal{E}) \) to optimality. If \( M \) is infeasible, **STOP** (the TD problem is infeasible)

2. Let \( t^* \) be the current optimal schedule. Find all line conflicts generated by \( t^* \).

3. If no conflicts exist
   
   3.a Solve the slave problem associated with \( t^* \). Let \( \bar{\mathcal{E}} \) be the violated combinatorial cuts.
   
   3.b If \( \bar{\mathcal{E}} \) is empty, **STOP** (\( t^* \) is the global optimal schedule)
   
   3.c Add \( \bar{\mathcal{E}} \) to \( \mathcal{E} \)

4. If some line conflicts exist
   
   Let \( \bar{\mathcal{O}} \) and \( \bar{\mathcal{F}} \) be the set of opposite and follower train pairs involved in the current line conflicts.

5. Add \( \bar{\mathcal{O}} \) to \( O \) and \( \bar{\mathcal{F}} \) to \( F \). GoTo 1

\(^7\)The timetable associated with a station \( s \) is simply the vector \( t^s \) of all arrival and departure times of the trains arriving and leaving the station.
In order to provide an initial upper bound to the master-slave algorithm we developed a simple, heuristic algorithm. In a similar fashion, we decompose the problem into a Line Dispatching problem and a Station Dispatching problem: however the master and the slave problems are solved by a heuristic procedure described in [25]. Moreover, a practical implementation of our heuristic algorithm is currently operating on several regional lines in Italy embedded in traffic management systems developed by Bombardier Transportation ([29]). As required by the operator, generated solutions are displayed to dispatchers, which can accept them or refuse them. Solutions are then carried out by a fully automatic route setting system which handles all aspects required to implement the solution, such as route settings, traffic lights and so forth.

A final interesting remark is that our decomposition and row generation approach mimics, in some sense, the actual behaviour of human dispatchers which detect potential conflicts and prevent them by establishing a suitable meeting point for the conflicting pairs. Dispatchers then force drivers to follow their decisions by switching traffic signals. Adding a violated constraint to the master is the mathematical equivalent of activating a red signal.

5.1 Warm-start enhancement

Dispatching trains is a (hard) task which is carried out repeatedly over time, generally with a tight window for decision making. In other words, dispatching trains involves solving a sequence $TD^1, TD^2, ...$ of hard optimization problems very fast over a period of time. During normal operations, the solution of the next problem in the sequence occurs a certain time after the previous has been solved, and is typically triggered by some event, such as a train approaching a station, a train leaving a station, a delay etc. The time between two successive solutions is strongly dependent on traffic conditions and can range from a few seconds in underground stations (see [30]) to several minutes in sparse, uncongested networks ([1]). In other terms, the faster the status of the network changes, the shorter the time between successive dispatching decisions. As a consequence, the MILPs (2) to (17) associated with two successive dispatching problems tend to be similar in terms of variables and constraints, i.e. they contain many variables and constraints which are associated with the same trains or train pairs, stations and tracks. Furthermore, we observed in our experiments that the optimal solutions of two successive problems are also, in many cases, "similar", that is normally many 0,1 variables will have the same value in the two solutions.

We may take advantage of these informal observations by initializing the master with constraints and variables from the previous problem in a warm-start fashion. Namely, when solving $TD^k$, at step 0 the subsets $O, F$ of the Master $M(O, F, E)^k$ are obtained from sets $O^{k-1}, F^{k-1}$ of the last master of the previous problem $TD^{k-1}$. Variables and constraints associated with pairs which cannot conflict anymore are removed.

Finally, optimization solvers often benefit from starting with an initial solution, even
if partial or infeasible. When using this warm-start feature, the values taken by the 0,1 variables of the previous optimal solution are provided in input when solving the next problem.

6 Computational Results

In this section we present computational results based on real-life instances which stem from our applications on some single- and double-track lines in Central and Northern Italy and from a single-track line in Norway. The purpose of our tests is manyfold. First we evaluate the effect of the enhancement described in Section 3.1. We show that the enhanced model is more effective both in terms of computational time and number of instances solved to optimality. Next, we compare solutions found using the new algorithm with the decisions carried out by dispatchers in Italy on the same instances, showing how the algorithm is able to significantly improve the current status-quo. We then show how introducing the "warm-start" feature (Section 5.1) further reduces computation time. Last, we present results based on real-life instances from the Dovrebanen line (Dombås - Trondheim) in Norway.

Details about the lines are given in Table 1, where "#Stops" stands for the total number of passenger stops on the line, "#Blocks" stands for the total number of block sections on the line, "Single" stands for Single-Track, "Mixed" stands for Single and Double-Track.

<table>
<thead>
<tr>
<th>Line</th>
<th>Country</th>
<th>#Stops</th>
<th>#Stations</th>
<th>#Blocks</th>
<th>Length (m)</th>
<th>Track types</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-B</td>
<td>Italy</td>
<td>14</td>
<td>21</td>
<td>95711</td>
<td>Single</td>
<td></td>
</tr>
<tr>
<td>O-T-F</td>
<td>Italy</td>
<td>25</td>
<td>53</td>
<td>283839</td>
<td>Mixed</td>
<td></td>
</tr>
<tr>
<td>D-T</td>
<td>Norway</td>
<td>4</td>
<td>36</td>
<td>209830</td>
<td>Single</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: T-B: Trento-Bassano, O-T-F: Orte-Terontola-Falconara, DT: Dombås - Trondheim.

Figures 1a, 1b show the layout of the O-T-F line, on which we have focused most of the experiments presented in this paper. The O-T-F line is centered in Foligno (Umbria) and presents, on one side, the Orte-Terontola branch, on the other, the Falconara-Orte branch.

Instances The instances in our experiments stem from the lines in Table 1 and refer to existing trains running at a specific instant in time, when a "snapshot" of the current status of the network (position of trains, delays etc.) was taken and saved. Italian

Typically commercial solvers are equipped with some method capable to adjust to some extent infeasible or partial solutions.

Two small parts of the line are not represented in these figures, namely Orte-Terni (3 stations, 1 passenger stop) and Montecarotto-Falconara (3 stations, 3 passenger stops)
instances were provided by the Italian infrastructure manager RFI ([35]), while Norwegian instances were provided by the Norwegian infrastructure manager Jernbaneverket ([23]). The scheduling and dispatching horizons were set to a minimum of 12 hours in all cases. The horizons were intentionally chosen to be large to present a greater challenge for the algorithm, but actually, in a practical setting, they can be chosen significantly smaller. A 60 second time-limit was fixed for the algorithm to find the optimal solution, a limit which is regarded as acceptable for dispatchers on such lines. This limit much depends on the congestion and the distance between stations, as discussed in Section 5.1.

**Objective function.** As indicated in Section 2, in agreement with the Railway operator we chose a convex, piecewise-linear function to model the objective. This function
includes terms which penalize the delay of each train in specific stations (normalized w.r.t. to their number and relative importance). Deviation from aimed arrival times is divided into different intervals with an increasing marginal cost. More specifically, the values chosen were: 1 for the segment between 0 and 5 minutes, 3 for the segment between 5 and 10 minutes and 9 for the half-line after ten minutes. These values were agreed with engineers at RFI and somehow reflect the significance of the various delay "categories". All trains were assumed to have the same relative priority (i.e. priority set to 1).

Implementation details. Test on Italian instances were run on a shared server with an Intel(R) Xeon(tm) E5620 CPU 2.40GHz 24G RAM. Tests on Norwegian instances were run on a Dell(R) laptop with an Intel(R) Core(tm) i7-2640M CPU 2.80GHz 8G RAM. In both cases, IBM ILOG CPLEX(R) 12.2 was used as solver. CPLEX parameters were set to default values. In the "warm-start" setting, CPLEX's MIP start function was used to input an initial solution.

6.1 Evaluating model enhancements

We now focus on evaluating the effectiveness of our model, and in particular the effect of the enhancement introduced in Section 3.1. In the following, we refer to the corresponding enhanced algorithm as Track Partitioning (TP), since the new variables are associated with tracks and in a sense the decisions taken by the algorithm partition the line (before and after the track). The non-enhanced algorithm is referred to as Station Meeting (SM) since conflict variables are associated with stations. In Table 2, we compare SM and TP in terms of solution quality and computation time for ten representative, real-life instances, selected in a large set from the T-B and O-T-F lines in Italy. Note that even though T-B is not a highly dense line, all connections are single track and trains are often running late. For each instance we show the number of trains in the dispatching horizon and how many of these are delayed (divided in three delay ranges: less than 5, between 5 and 10, and more than 10 minutes delay).

Results show how TP is able to solve all instances in Table 2 to optimality (0% gap) within the time limit. Moreover, in the first 7 cases both models are able to solve to optimality, though TP in no case requires more than 9 seconds to find the optimal solution. In the last 3 instances, SM is not able to find the optimal within 60 seconds. The relative gap to optimality was in two cases around 38%, decreasing decidedly in the last case (the largest instance in the test-bed) to over 60%. Although not exhaustively, these instances indicate a trend. Not surprisingly, computation time is generally affected by line topology, number of trains and number of delayed trains. This is particularly evident for the SM model. Clearly, other factors also account for some of the variability in computation time, erraticism in the branch and bound process above all. In any case, the results on the 10 instances presented in Table 2 provide a fair representation of the overall behavior of the two models on the whole set of test
instances.

### 6.2 Comparing with the current practice

A decision support system based on a heuristic version of our decomposition approach has been in operation on the Trento-Bassano, Orte-Terontola-Falconara and other lines in Italy since 2011 ([29]). Embedded in a Traffic Management System, the algorithm computes in real-time one or more solutions to the current dispatching problem. Dispatchers visualize the status of the network on a screen which displays a train graph, where, in a typical fashion, space and time are represented on the vertical and horizontal axis, respectively. A snapshot of such train graph is shown in Figure 2. A list of open conflicts is presented, each with different solution options which are ordered in terms of heuristic criteria (previously agreed with the practitioners). At this point, the dispatcher decides whether to accept the solution proposed by the system or choose a different one. After an initial period of accustomization, figures show that dispatchers accept the first solution around 93% of times.

In this section, we compare the solutions of our exact approach with the decisions carried out by dispatchers on the Orte-Terontola-Falconara line during a day in January 2013. The average number of controlled trains (see Section 2) on the O-T-F line on this day was 86, with a minimum of 51, a maximum of 130 and a standard deviation of 27. The average number of trains running late was 5.

Diagram 3a shows average train density for the O-T-F line. In red, "Total Trains" refers to the total number of controlled trains in the dispatching horizon, that is, the
Figure 2: Train graph displayed in dispatching center with interactive box. Lines identify trains, green for passenger, purple for freight and red for work trains, while circles identify conflicts, white for those solved and light blue for those still to be handled. The pop-up box in the right hand corner shows the interaction between system and dispatchers.

(a) Train density  
(b) Comparing delay ranges

Figure 3

total number of trains taken in account by the algorithm when solving. In blue, "Online Trains" refers to the number of trains which are physically on the controlled line at the given time. To highlight the impact of dispatching decisions, in Table 3 we also introduce a natural performance indicator, namely train punctuality. This measure is immediately understandable both by railway practitioners and by the general public, and it is indeed what dispatchers try to improve by their decisions.

In Table 3 and Diagram 3b, we report the average distribution of delayed trains for TP against the solutions which were carried out by dispatchers, computed by solving 5714 instances taken at different times of the day (around 4 times a minute). Based on feedback from the railway operators, possible delays were subdivided in four macroscopic ranges: on-time (less than 5 minute delay), delays between 5 and 10 minutes, delays between 10 and 15 minutes and delays greater than 15 minutes. Trains were
then clustered based on the difference between expected and actual arrival time at destination.

<table>
<thead>
<tr>
<th>Decisions</th>
<th>On Time</th>
<th>5&lt;delay≤10</th>
<th>10&lt;delay≤15</th>
<th>delay&gt;15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchers</td>
<td>84.7%</td>
<td>1.8%</td>
<td>1.4%</td>
<td>12.1%</td>
</tr>
<tr>
<td>Exact method</td>
<td>90.9%</td>
<td>4.1%</td>
<td>1.8%</td>
<td>3.2%</td>
</tr>
</tbody>
</table>

Table 3: Average punctuality distribution

Results show a significant increase in the number of trains on time (6.2%), with a consequent decrease of trains in the less desirable ranges.

<table>
<thead>
<tr>
<th>Periods</th>
<th>Iterations</th>
<th>Conflicts</th>
<th>Initial</th>
<th>Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Potential</td>
<td>Solved</td>
<td>Rows</td>
</tr>
<tr>
<td>[04:08]</td>
<td>9</td>
<td>13777</td>
<td>154</td>
<td>3228</td>
</tr>
<tr>
<td>[08:12]</td>
<td>9</td>
<td>11639</td>
<td>151</td>
<td>2955</td>
</tr>
<tr>
<td>[12:16]</td>
<td>9</td>
<td>7056</td>
<td>126</td>
<td>2287</td>
</tr>
<tr>
<td>[16:20]</td>
<td>5</td>
<td>3164</td>
<td>49</td>
<td>1543</td>
</tr>
<tr>
<td>[20:24]</td>
<td>5</td>
<td>3617</td>
<td>61</td>
<td>1651</td>
</tr>
</tbody>
</table>

Table 4: Algorithmic information

In Table 4 we also present algorithmic information for instances used in Table 3. In particular, we aggregate the instances in periods of the day (Periods), and present (average) information regarding: number of macro-iterations of the algorithm (Iterations, see Section 5), initial number of rows and variables in MILP (Initial), number of potential and solved line conflicts (Conflicts), (generated) rows and variables in MILP associated with solved conflicts (Generated). "Conflicts" give a measure of the advantage given by using a delayed row and column generation approach. In fact, in each period, at most 2% of the total number of "potential" conflicts\(^{10}\) are, on average, actually solved (i.e. variables and constraints associated with these conflicts are added to the MILP). Avoiding to add variables and constraints associated with the remaining 98% allows us to tackle a significantly "lighter" MILP, which in instances of practical interest may mark the line between being and not being able to find solutions to TD within the time limit.

\(^{10}\)By potential conflicts we refer to the total number of line conflicts which may in principle occur between any given pair of trains
6.3 Testing the warm-start approach

In Subsection 5.1 we introduced an idea to enhance the algorithm's performances in a realistic setting based on exploiting previous solutions (warm start). In Table 5 we compare the performance of the algorithm with or without using the warm-start feature. Time(s) indicates the time required to find the optimal solution (seconds as time unit), Runs indicates the number of distinct runs which were performed, # Trains indicates the average number of controlled trains on those days. These tests were run on the same test-bed from which the instances in Table 2 were extracted (T-B and O-T-F lines). However, here we consider all instances extracted over the course of the day, namely 803 instances for T-B, with around 100 seconds time lap between successive instances, and 5714 instances for O-T-F, with around 15 seconds of time lap.

| Line | Runs | # Trains | Warm start | % Optimal | Time(s) |  |  |
|------|------|----------|------------|-----------|---------|  |  |
|      |      |          |            |           | Mean    | Std Dev |
| T-B  | 803  | 27       | No         | 87.3%     | 4.88    | 9.96  |
| T-B  | 803  | 27       | Yes        | 92.0%     | 2.30    | 5.27  |
| O-T-F| 4617 | 86       | No         | 99.7%     | 14.11   | 10.22 |
| O-T-F| 4617 | 86       | Yes        | 97.9%     | 3.49    | 2.68  |

Table 5: Comparing algorithm runs with (without) the warm-start feature.

For both lines, results show an improvement of average computation time with (small) variations in terms of instances for which optimality was found. In particular: for the T-B line, average computation time was significantly reduced by around 50% (4.88 to 2.30 seconds) with a decrease in variability (standard deviation from 9.96 to 5.27), while increasing the number of instances solved to optimality within the time limit (from 87.3% to 92%); for the O-T-F line, average computation time was reduced by an impressive 75% (14.11 to 3.49 seconds) with a large decrease in variability (standard deviation from 10.22 to 2.68). In the latter case, the number of instances solved to optimality within the time limit was, however, slightly reduced (from 99.7% to 97.9%). An intuitive explanation of the improvements (in terms of computation time) can be attributed to the nature of the algorithm. As shown in Section 5, when tackling the master problem we iteratively solve a series of MILPs to optimality, verify if the current optimal solution is feasible and, if not, accordingly separate suitable constraints. This process may, in some cases, be relatively time consuming, as the number of MILPs may grow with the size of the problem. However, in this warm-start scheme, the sets of constraints (and associated variables) which have a high chance of being added to the problem at some iteration of the master, are instead added to the master beforehand, allowing the algorithm to "skip" the iterations required to identify such conflicts. Furthermore, the algorithm is provided with an initial solution. Now, in principle, this solution may not be feasible in the current problem. However, due to the "local stabil-
ity" of optimal solutions discussed in the previous section, we observed that in many cases, when feasible, such solution will be close to the optimal\(^\text{11}\). In other cases, it may provide a good upper-bound to the optimum of the current problem. Furthermore, using repair heuristics and similar features, CPLEX can still benefit from starting with an infeasible solution.

In conclusion, our results show that such feature allows to tackle the train dispatching problem in a real-life setting (when solving dispatching problems repeatedly over time) even more effectively. Indeed, using the warm-start feature may prove to be crucial, particularly when controlling highly congested lines where the required response time for decision making is very stringent.

### 6.4 Tests on Norwegian instances

We tested our algorithm on instances stemming from a project with the Norwegian infrastructure manager (Jernbaneverket), which were extracted from the Dovrebanen line (Dombås-Trondheim) in the fall of 2013, namely a day in September and a day in October. In Table 6, we summarize results obtained by running \(TP\) on such instances, where \(\text{Time(s)}\) indicates the time to find the optimal solution (time unit is seconds), \(\text{Runs}\) indicates the number of distinct runs which were performed, \# \(\text{Trains}\) indicates the average number of controlled trains and \# \(\text{Late}\) indicates the average number of late trains.

<table>
<thead>
<tr>
<th>Date</th>
<th>Runs</th>
<th># Trains</th>
<th># Late</th>
<th>% Optimal</th>
<th>Time(s)</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>17/09/13</td>
<td>728</td>
<td>38</td>
<td>4</td>
<td>100%</td>
<td>1.17</td>
<td>1.17</td>
<td>1.00</td>
</tr>
<tr>
<td>01/10/13</td>
<td>260</td>
<td>47</td>
<td>6</td>
<td>100%</td>
<td>4.45</td>
<td>4.45</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Table 6: Results from the Dovrebanen (Dombås-Trondheim line) in Norway.

As indicated in Table 6, 728 runs from the 17th of September were all solved to optimality within just over a second (on average). Later, 260 runs from the 1st of October were performed. The 1st of October proved to be a slightly more trafficked day (47 average controlled trains instead of 38) and average computation time was slightly higher (4.45 seconds) but still significantly below the acceptable time limit in this area ([1]). These results have been reviewed in detail and the quality of proposed solutions was further confirmed by dispatchers from the Marienborg central (Trondheim).

\(^{11}\text{At least for those components which are still in the problem}\)
7 Conclusion and future challenges.

In this paper we present an exact methodology to tackle the Train Dispatching problem and test it on real-life instances which stem from our applications. Furthermore, we introduce a novel "partitioning" approach for line conflicts, a pure IP model for train platforming and a detailed description of the solution algorithm for TD. We show that this approach can be successfully applied to dispatch small-medium lines (e.g. regional lines). On the other hand, we do not handle complex rail nodes (e.g. a sub-network of several interlinked lines with multiple parallel, bi-directional tracks) or in general, networks not satisfying the assumptions in Section 4. Thus, some developments are required to extend the approach to handle any railway line. These authors believe that such extension can be obtained exploiting one of the decomposition's main advantages: substitution of one model for another. So, for instance, developing efficient (not necessarily exact) approaches to manage complex rail nodes could allow to tackle such lines.

References


[23] Jernbaneverket, the Norwegian government’s agency for railway network services. www.jernbaneverket.no.


