An exact decomposition approach for the real-time Train Dispatching problem

Article in Operations Research · January 2015
DOI: 10.1287/opre.2014.1327

CITATIONS
7

READS
34

2 authors:

Leonardo Lamorgese
Optrail
12 PUBLICATIONS 32 CITATIONS

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

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

SESAR WP-E ZefMap 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.
An exact decomposition approach for the real-time Train Dispatching problem

Leonardo Lamorgese * Carlo Mannino *

Abstract

Trains movements on a railway network are regulated by official timetables. Deviations and delays occur quite often in practice, demanding fast re-scheduling and re-routing decisions in order to avoid conflicts and minimize overall delay. This is the real-time train dispatching problem. In contrast with the classic "holistic" approach, we show how to decompose the problem into smaller sub-problems associated with the line and the stations. This decomposition is the basis for a master-slave solution algorithm, in which the master problem is associated with the line and the slave problem is associated with the stations. The two sub-problems are modeled as mixed integer linear programs, with specific sets of variables and constraints. Similarly to the classical Benders' decomposition approach, slave and master communicate through suitable feasibility cuts in the variables of the master. Extensive tests on real-life instances from single and double-track lines in Italy showed significant improvements over current dispatching performances. A decision support system based on this exact approach has been in operation in Norway since February 2014, and represents one of the first operative applications of mathematical optimization to train dispatching.

1 Introduction

In a first, general picture, a rail network may be viewed as a set of stations connected by tracks. Each train runs through an alternating sequence of stations and tracks (train route). Each route also includes the (possibly complicated) movements performed by a train within each station. Trains run along their routes according to the production plan; the latter specifies the movements (routing) and the times when a train should enter and leave the various segments of its route (schedule), including station arrival and departure times, which define the official timetable. The generation of the production plan is typically decomposed into two successive phases. In the first phase a tentative official timetable is established and the arrival and departure times are fixed. In the
second phase, called *train platforming* or *track allocation* (see [7], [11]) the complete routes (including station movements) for trains are established, sometimes by allowing moderate deviations from the tentative timetable.

With some exceptions, the production plan ensures that no two trains will occupy simultaneously incompatible railway resources (*conflict free schedule*) \(^1\). However, when actually running, one or more trains can be delayed and potential conflicts in the use of resources may arise. As a consequence, re-routing and re-scheduling decisions must be taken in real-time. These decisions are still, in most cases, taken by human operators (*dispatchers*), and implemented by re-orienting switches and by controlling the signals status, or even by telephone communications with drivers. Dispatchers take such decisions trying to alleviate delays, typically having in mind some train ranking or simply following prescribed operating rules. What dispatchers implicitly do is solve an optimization problem - and of a very tough nature. Following [34], we refer to this problem as the *real-time Train Dispatching* problem (RTD).

In short, the RTD amounts to establishing, for each controlled train and *in real-time*, a route and a schedule such that no conflicts occur among trains and some measure of the deviation from the official timetable is minimized. As such, the RTD falls into the class of *job shop scheduling problems*, where trains correspond to *jobs* and the occupation of a railway resource is an *operation*. Two alternative classes of formulations have been extensively studied in the literature for job shop scheduling problems and consequently applied to train scheduling and routing problems: *time indexed formulations* [17] and *disjunctive formulations* [4].

In time indexed formulations (TI) the time horizon is discretized, and a binary variable is associated with every operation and every period in the time horizon. Conflicts between operations are prevented by simple packing constraints. Examples of applications of (TI) to train timetabling can be found in [7], [8], [10], [11], [20], [34], [40]: actually the literature is much wider, and we refer to [9, 24] for extended surveys. To our knowledge, basically all these works deal with the track allocation problem, which is solved off-line and where the number of time periods associated with train routes is reasonably small. In contrast, in the RTD the actual arrival and departure times may differ substantially from the wanted ones. Consequently, the number of time periods grows too large to be handled effectively by time-indexed formulations within the stringent times imposed by the application, as extensively discussed in [26]. Another drawback with (TI) formulations is that, if the time step is not chosen carefully, they may easily lead to solutions which are practically unattainable (see [20]).

In disjunctive formulations, continuous variables are associated with the starting times of operations, whereas a conflict is represented by a disjunctive precedence constraint, namely, two standard precedence constraints one of which (at least) must be satisfied by any feasible schedule. A *disjunctive graph* ([3]), where disjunctions are rep-

---

\(^1\)The problem of designing optimal production plans is of crucial relevance for railway operators. As pointed out in [24] "optimum resource allocation can make a difference between profit and loss for a railway transport company"
resented by pairs of directed arcs, can be associated to such disjunctive formulation and its properties can be exploited in solution algorithms. This type of disjunctive formulations can be easily casted into mixed integer linear programming models by associating a binary variable with every pair of (potentially) conflicting operations and, for any such variables, a pair of big-M precedence constraints representing the original disjunction. These constraints contain a very large coefficient and tend to weaken the overall formulation, which is the main reason why (TI) formulations were introduced.

The connection between railway traffic control problems, job shop scheduling and corresponding disjunctive formulations was observed quite early in literature. However, a systematic and comprehensive model able to capture all relevant aspects of the RTD was described and studied only in the late 90s by Mascis ([27]) and further developed in [28]. In these works, the authors also introduce a generalization of the disjunctive graph to cope with this class of problems. After these early works there has been a flourishing of papers representing the RTD by means of disjunctive formulations and exploiting the associated disjunctive graph. Recent examples can be found, e.g., in [15] and [38]. The great majority of these papers only use disjunctive formulations as a descriptive tool and eventually resort to purely combinatorial heuristics to solve the corresponding RTDs, as in [37]. The explicit use of the disjunctive formulation or their reformulations as mixed integer linear programs (MILPs) to compute bounds is quite rare, and typically limited to small or simplified instances. Examples are [26], which handles small-scale metro instances, and [38], which introduces several major simplifications, drastically reducing the instances’ size.

So, mixed integer linear programming is rarely applied to solve real life instances of the RTD: time-indexed formulations tend to be too large and often cannot even produce a solution within the time limit; big-M formulations tend to be too weak and can also fail to produce feasible solutions within the time limit. Actually, the lack of real life implementations of the many theoretical studies affects all known approaches, exact or approximated, as recently observed in [9, 20, 31]. In [20] the author conjectures that the application of optimization to regular dispatching activities is imminent: the work presented in this paper confirms his conjecture.

Indeed, we introduce a new modeling approach for the RTD and a solution methodology which allows to overcome some of the limitations of natural big-M formulations and solve to optimality a number of real life instances in single- and double-track railway lines within the stringent time limits imposed by the application. The methodology is based on a decomposition of the RTD into two sub-problems. The first, called the Line Dispatching problem (LD), amounts to establishing (in real-time) a timetable that minimizes the deviation from the official one while ensuring that trains never occupy simultaneously incompatible line tracks. The second further decomposes into several, independent subproblems, one for each station, called Station Dispatching problems (SD). The SD is the problem of routing and scheduling trains in a station according to a given timetable. The LD and each SD give raise to distinct sets of variables and constraints. Our approach resembles the classical Benders’ decomposition or, more
precisely, its combinatorial variant introduced by Codato and Fischetti in [14]. In our
decomposition, the LD acts as the master problem, whereas the SD is the slave. The
LD is defined on a simplified network, in which each station is represented by a node,
and is solved exactly. The solution of the LD produces, for each train, tentative arrival
and departure times in the stations of the railway line. The slave problem is a feasibility
problem and amounts to finding, for each train, a route in each station which is compati-
ble with the tentative arrival and departure times and is conflict-free. Similarly to [14],
if the slave problem is infeasible, then a violated (combinatorial) cut in the variables of
the master problem is 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 sub-problem is then rather small and can be easily
solved.

The decomposition has two major advantages. First, the number of variables and
big-M constraints is drastically reduced with respect to the big-M formulation. Second,
depending on the specific infrastructure, we may choose different models to represent
the stations in the SDs. As we will show in Section 4, the (general) SD is NP-hard.
However, in some cases of practical impact, simpler models can be exploited, leading
to polynomial cases. One such case (occurring in our real-life instances) is described in
Section 4 along with two different solution approaches. Actually, since the lines may
contain quite different station layouts, different models can be applied simultaneously.
Also, one can start by using the simplified version in every station of the line, refining
the model only if a violation of the associated constraints occurs. Interestingly, this
decomposition resembles the normal practice of railway engineers to distinguish between
station tracks and line tracks and of actually tackling the two problems separately.
However, the master-slave scheme allows us to find globally optimal solutions.

An implementation of our exact decomposition approach has been in operation
in Norway since February 2014, supporting dispatchers in Stavanger control center
by presenting solutions to the RTD. The system was developed by SINTEF ([36]),
the largest independent research institute in Scandinavia, supported by the Norwegian
network operator (JVB[22]) and train companies (NSB[29], Flytoget[18], CargoNet[13]).
Due to the positive feedback and results so far, stakeholders are planning to extend the
system to other dispatching areas in Norway.

Moreover, the algorithm has also been largely tested on instances from Italian lines,
showing significant improvements with respect to performances of the currently oper-
ative traffic control system developed by Bombardier Transportation². This system,
operating in a number of single and double-track lines in Italy since 2011, also exploits
our decomposition approach to find dispatching solutions. However, for the time being,
this implementation does not use the algorithm to its full extent as sub-problems are
solved heuristically. In addition, the LD and SDs sub-problems are solved independ-
ently, which can result in infeasible solutions.

Both implementations in Italy and Norway represent remarkable practical achieve-

²One of the largest multinational transportation companies
ments. In fact, according to two 2014 studies [9, 31], there are no other optimization based dispatching systems currently in operation.

In sections 2 to 5 we describe the main modeling and algorithmic ingredients of our approach. In our description, we have chosen to make some simplifying assumptions and omit minor details, as they do not give the reader further insight in understanding the methodology. However all these aspects were (inevitably) taken into consideration in the real-life implementations (see Section 7).

Summarizing the major contributions of this paper to the current practice:

- We introduce an exact decomposition approach to the real-time Train Dispatching problem.
- We give some complexity results on variants of the sub-problems in the decomposition which are relevant to the practice.
- We show how to effectively model the sub-problems by mixed integer programs and how to apply delayed row generation to couple them.
- We describe the implementation of a decision support system based on our exact approach which is currently in operation in Norway.
- We show that such exact approach can significantly improve current dispatching performances for some lines in Italy.

2 The dispatching problem

A Railway Network is a set $S$ of stations and a set $B$ of tracks (called blocks) connecting pairs of stations. Blocks are often partitioned into sections, and, for safety reasons, trains running in the same direction on the same block will be separated by (at least) a fixed number of such sections. We neglect sections in the remainder of the paper but extending the model to handle such case is immediate. We also neglect other railway infrastructures, such as sidings and cross-overs, but again the extension is straightforward. Similarly, safety constraints can be easily modeled, but we do not discuss them here. Next, we examine the elements of the railway network.

**Stations.** A station can be viewed as a set of track segments, the minimal controllable rail units, which in turn may be distinguished into stopping points and interlocking-routes. A stopping point is a track segment in which a train can stop to execute an operation. Two special stopping points are those associated with the entrance and the exit to the station. An interlocking-route is the rail track between two stopping points, and is actually formed by a sequence of track segments. For our purposes, a station $s \in S$ is represented by means of a directed graph $G(s) = (N_s, E_s)$ where $N_s$ is the set of stopping nodes (corresponding to points) and $E_s \subseteq N_s \times N_s$ is the set of interlocking
arcs (corresponding to routes). A train going through a station $s$ is running a directed path in the station graph. The path usually contains a platform node, where a train can, if required, embark or alight passengers. Also, if the train enters (exits) the station, the path will contain an entrance (exit) node.

In Figure 1 we give the classical schematic representation of a station along with the associated graph. The three platforms correspond to nodes 2, 3, and 4 in the graph, whereas nodes 0 and 1 are the incoming and exit nodes.

![Figure 1: From the station scheme to the oriented graph. Nodes 2, 3, and 4 correspond to platforms.](image)

**Trains and routes.** Our purpose is to model the real-time movements along the line of the set $T$ of controlled trains. Some of the trains will not have entered the line yet, while others will be in stations or running on tracks between stations. A train $i \in T$ running through the line from an initial position to the destination station will traverse all intermediate stations and blocks. We represent this movement by a graph $R(i) = (V^i, A^i)$ called train route. The nodes of $R(i)$ are associated with blocks, with (station) stopping nodes and with (station) interlocking arcs traversed by train $i$. Graph $R(i)$ is a directed simple path. Every arc $(u, v) \in A^i$ has a weight $W_{uv}$, and represents a simple precedence constraint, i.e. $v$ is encountered by train $i$ right after $u$, with $W_{uv}$ being the minimum time to move from $u$ to $v$. So, if $u$ is the block connecting station $A$ to station $B$ then $v$ is the entrance node of station $B$ and $W_{uv}$ is the minimum running time\(^3\). If $u$ is a platform in a station then $v$ is an interlocking arc leaving $u$ and $W_{uv}$ is the time spent to embark and alight passengers, etc. Since $R(i)$ is a directed path, its nodes are naturally ordered and we let $V^i = \{v^i_1, v^i_2, \ldots \}$. Every route will also include an artificial node (the last) representing the out-of-line state. In Figure 2 we show one such route for a train $i$. Circle nodes correspond to tracks preceded by signals, whereas diamond nodes are interlocking routes or tracks between stations.

\(^3\)In this presentation we assume fixed speed profile.
Figure 2: A train route. Circle nodes correspond to tracks preceded by signals, whereas diamond nodes are interlocking routes or tracks between stations.

The real-time schedule. We now consider a new graph $R = (V, A)$, referred to as the graph of routes, which is the union of all route graphs $R(i), i \in T$ plus an additional vertex $O$ (the Origin) and a directed arc from $O$ to the first node $v^i_1$ of each route $R(i)$ with $i \in T$. Each of these arcs has a weight $W_{Ov^i_1}$ assigned to it which equals the expected time for train $i$ to start its route.

Every node $v$ in graph $R$ (except the origin) represents the occupation of a rail resource by some train. With every node $v$, we associate a non-negative continuous variable $t_v$. For $v \in V \setminus \{O\}$ and $v = v^i_k$, the quantity $t_v$ represents the (earliest) time in which train $i$ can enter the $k$-th node on its route, i.e. the time when the corresponding rail resource can be occupied by train $i$. Also, we let $t_O = 0$: in other words, node $O$ represents the start of the planning time. Vector $t \in \mathbb{R}^+_V$ is called real-time schedule. Clearly, every feasible schedule must satisfy the following set of precedence constraints:

$$t_v - t_u \geq W_{uv} \quad (u, v) \in A \quad (1)$$

Other simple precedence constraints may be easily represented on the graph of routes. For instance, the official departure time $D_{is}$ of train $i$ from station $s$ is a lower bound on its actual departure time, and can be represented by an arc, with weight $D_{is}$, from the origin $O$ to the exit node of $i$ from $s$.

Any feasible schedule is such that no two trains occupy simultaneously the same rail resource or incompatible ones. So, let $i, j \in T$ be distinct trains, let $v^i_k, v^j_l \in V$ and assume that the rail resource corresponding to $v^i_k$ and the rail resource corresponding to $v^j_l$ cannot be occupied simultaneously. In other words, either train $i$ enters next rail resource $v^i_{k+1}$ on its route before $j$ enters $v^j_l$, or train $j$ enters next rail resource $v^j_{l+1}$ before $i$ enters $v^i_k$. This can be expressed by the following disjunctive constraint:

$$(t_{j,l} - t_{i,k+1} \geq \epsilon) \lor (t_{i,k} - t_{j,l+1} \geq \epsilon) \quad (2)$$

where we let $t_{x,y} = t_{vx}$ to simplify the notation, and where $\epsilon$ is a suitable positive constant to represent interaction with the infrastructure. There is one such constraint for every pair of incompatible rail resources visited by any two distinct trains. Disjunctions
of precedence constraints are represented in graph drawings by pairs of dotted arcs. In Figure 3 we show a graph of routes with two routes and two disjunctive precedence constraints.

**Figure 3:** Graph $R$ with disjunctive constraints. Train A cannot enter the track between stations 1 and 2 before the train B has entered Station 2, or vice versa.

**Objective Function** The quality of the real-time schedule $t$ depends on its conformity to the official timetable. Following initial discussion with engineers from Bombardier Transportation and the Italian network operator we adopted a convex, piecewise linear function. Remarkably, such function also shapes the indications received by the dispatchers in Stavanger control center (Norway). Let $a^i_s$ be the arrival time of train $i \in T$ in station $s \in S$. We associate with $a^i_s$ the delay cost function $c^i_s$ depicted in Figure 4. The cost for a train is obtained by summing up the delay costs in every station of its route and the overall cost $c(t)$ is the sum of the costs of all trains.

**Figure 4:** The cost function agreed with practitioners. For train $i$ and station $s$, $W^i_s$ is the official arrival time, whereas $a^i_s$ is the actual arrival time.
The real-time Train Dispatching problem  We are now able to state the RTD 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 and an associated real-time schedule satisfying all of the (simple) precedence constraints (1) and all of the disjunctive precedence constraints (2) so that the cost function $c(t)$ is minimized.

Remark that, in order to solve the RTD, we need to solve both a routing and a scheduling problem. The RTD can be easily modeled by Mixed Integer Linear Programming (MILP) formulations (as in [26]) or some other techniques to tackle disjunctive programs. However, RTD instances of practical interest are typically so large that corresponding MILP models cannot be solved by direct invocation of a commercial solver or by applying standard solution techniques. For this reason most authors resort to heuristic approaches or to simplified versions of the problem.

We have followed a different path, namely we developed a decomposition technique which makes it possible to apply classical MILP techniques and solve to optimality instances of the RTD of practical interest. In what follows, we start by discussing the case of single-track lines. As we will show this is already an interesting problem per se, and it allows us to easily introduce the basic concepts of our decomposition approach. The extension to double-track lines is straightforward and will be discussed at the end of Section 3. The case of more complex line layouts (i.e. lines with cross-overs) is not addressed in this paper.

Single-track lines  "Single-track" means that there is only one track between any two stations. Single-track lines still play a central role in the global railway transportation system. Indeed, a vast majority of the railway system is, at the present day, still single-track. For example, in December 2011, in Italy there were 9218 km of single-track lines against 16723 km in total ([32]). Italy is not an exception in the European Union. According to the UIC (*Union Internationale des Chemins de fer*, i.e. the worldwide union of Railway Operators), a number of countries including Spain (65% single-track), France (45%) and Germany (46%) also have a large share of single-track railways [39]. On a more global scale, some of the world’s largest and fastest growing economies like the Russian federation, China and India also present very significant figures in this sense. In Russia, 47748 km out of 85167 (56%) are single-track [39], while the proportion is even more impressive in China (40257 km out of 66050, or 61%) [39] and India (45237 km out of 64460, or 70%) [21]. Aside from being amongst the countries with the highest single-track ratio in the world, China and India rank second and first, respectively, in terms of rail usage statistics, with a staggering 816 and 978 billion passenger-km share, contributing almost entirely to the Asia and Oceania quota, which represents 75% of the total worldwide [39]. Overall, according to [39], in 2011 single-track lines represented ca. 80% of worldwide railway system. Clearly, the share of passengers transported on
single-track lines may be smaller. Still, these figures indicate unequivocally that the RTD for single-track railways represents a (hard) problem relevant to the practice, with global social and economical impact.

In single-track lines, stations in $S = \{1, \ldots, q\}$ are connected by single-tracks (blocks), with block $i$ joining station $i-1$ and station $i$. Observe that, in this situation, the routing problem is only limited to stations, as there is only one way to go from a station to another. Also, if two trains meet somewhere on the line, this must happen in a station (or some similar facility).

As discussed in the introduction, we decompose the RTD into the real-time Line Dispatching problem (LD), which amounts to establishing a schedule for the trains so that they only meet in stations (or they do not meet at all), minimizing a given cost function; and into a feasibility problem which further decomposes in a number of independent (real-time) Station Dispatching problems (SD). Each SD is a feasibility problem which amounts to finding suitable routes and a schedule in a station which matches a given timetable. Again driven by our application, we will consider only small sized stations, with some important consequences on the adopted models. The two sub-problems in the decomposition are not independent of each other. In fact, a solution to the LD may result in an inadmissible timetable for one or more of the SDs, as we may not be able to assign station routes to trains as scheduled by the LD (for example when the number of trains simultaneously in the station exceeds the number of platforms available). We later show how to re-couple the problems in the decomposition through a suitable master-slave solution mechanism.

3 Modelling the Line Dispatching problem

The first problem we discuss is the real-time Line Dispatching (LD) Problem. We conventionally extend the line with two additional fictitious stations, one for each side, able to accommodate any number of trains. Trains meeting in one of these stations are interpreted to meet outside the line. As in this sub-problem we are neglecting train movements within stations, we handle simple routes. In particular, for each train $i$, its route is an alternating sequence of stations and blocks and can be represented by the simple directed path $R(i) = \{(v_i^1, v_i^2), (v_i^2, v_i^3), \ldots, (v_i^{l(i)-1}, v_i^{l(i)})\}$ where node $v_k^i \in R$ for $1 \leq k \leq l(i)$ is either a station or a block. The last node $v_{l(i)}^i$ is always the destination station, whereas the first node $v_1^i$ may be a station or a block, depending on the position at time 0 of the train on the line. If $v_k^i$ is a station then $v_{k+1}^i$ is the next block on the route of train $i$, and the weight of the arc $(v_k^i, v_{k+1}^i)$ is the minimum time the train is supposed to spend in the station. If $v_k^i$ is a block then $v_{k+1}^i$ is the next station on the route of train $i$, and the weight of arc $(v_k^i, v_{k+1}^i)$ is the minimum running time of the train through the block. Particular care must be taken for the first arc $(v_1^i, v_2^i)$, where the weight represents a residual time.

Once again we can consider a set of trains $T$ running through the line, the corresponding graph of routes $R = (V, A)$, obtained as described in Section 2, and the
associated schedule $t \in \mathbb{R}_+^V$. The schedule $t$ approximates the behaviour of trains along the line. In this simplified setting, if $v$ is a node representing station $s$ on the route of train $i$, then $t_v$ is the arrival time of train $i$ in station $s$. Similarly, if $v$ is a node representing the block outgoing a station $s$ on the route of train $i$, then $t_v$ is the exit time of train $i$ from station $s$. Since we are dealing with small stations, this time closely approximates the train’s departure time from the station\footnote{Indeed, we are neglecting possible differences in running time between alternative paths within the station from the platform to the track. By slightly complicating the following station model these times can be made exact.}. Official departure times are of course lower bounds on actual departure times, and can be immediately represented by weighted arcs from the origin to the nodes representing stations.

Consider now two distinct trains $i$ and $j$ and let $R(i)$ and $R(j)$ be their respective routes. Assume that the trains meet in station $s \in S$ and let $v^i_k$ and $v^j_m$ be the nodes representing station $s$ on route $R(i)$ and $R(j)$, respectively. To simplify the following discussion, we may assume neither of these nodes is the last on its route.

Now, since the two trains meet in $s$ then train $i$ exits station $s$ after train $j$ arrives in $s$ and train $j$ exits station $s$ after train $i$ arrives in $s$. In other words, the schedule $t$ must satisfy the following (conjunctive) pair of constraints:

\begin{align}
  t_{i,k+1} - t_{j,m} & \geq \epsilon \\
  t_{j,m+1} - t_{i,k} & \geq \epsilon
\end{align}

where $\epsilon$ is a positive constant which depends on the infrastructure. Observe that the above precedence constraints correspond to adding to graph $R$ the arcs $(v^j_m, v^i_{k+1})$ and $(v^i_k, v^j_{m+1})$, with weight $\epsilon$. This is depicted in Figure 5, where we consider the case of two trains running in opposite directions and meeting in station $s_5$; the two precedence constraints are represented by arcs.

![Figure 5](image_url)

**Figure 5:** Two trains running in opposite direction and meeting in station $s_5$. The movements satisfy the two precedence constraints represented on the graph by two directed arcs.
In the following, trains \( i \) and \( j \) running in the same direction will be referred to as *followers*, and as *crossing trains* otherwise. To simplify the discussion we assume now that trains will meet at most once on the line. This is obvious for crossing trains, but not true in general for a pair of followers, even though this is almost always the case in practice. Once again, this assumption can be easily dropped by a straightforward extension of the model. Another assumption we make for followers is that when the following train catches up with the other train, it becomes the leading train after the meeting (the so called *pass event*). This is what typically happens in practice, where the train catching up is a faster one; also this assumption can be easily dropped in a slightly extended model.

Consider now a pair of followers \( i \) and \( j \) and assume that \( i \) precedes \( j \) before they meet in \( s \) and \( j \) precedes \( i \) after the meet. Let us assume that the trains meet in station \( s \in S \) and let \( v_{i}^k \) and \( v_{j}^m \) be nodes representing station \( s \) on route \( R(i) \) and \( R(j) \), respectively. Then schedule \( t \) will satisfy constraints (3) and (4). In addition, since we are considering single section tracks, for safety rules the following train cannot enter a given block before the leading train has left it, i.e. it has entered the next station on the block. Since \( i \) is leading before station \( s \) and \( j \) after, safety constraints can be expressed by the family of constraints \( t_{j,m-1} - t_{i,k} \geq \epsilon, t_{j,m-3} - t_{i,k-2} \geq \epsilon, \ldots (i \text{ leading before station } s), \) and \( t_{i,k+1} - t_{j,m+2} \geq \epsilon, t_{i,k+3} - t_{j,m+4} \geq \epsilon, \ldots (j \text{ leading after station } s). \) Correspondingly, we may represent these constraints on the graph of routes by the set of arcs \( A_{ij}^s = \{(v_i^k, v_j^{m-1}), (v_i^{k-2}, v_j^{m-3}), \ldots, (v_i^{m+2}, v_j^k), (v_i^{m+4}, v_j^{k+3}), \ldots \} \), as shown in Figure 6.

![Figure 6: Precedence constraints for two followers meeting in s5, represented by arcs on the graph of the routes.](image)

So, in general, the meeting condition of train \( i \) and train \( j \) in station \( s \) translates into a family of precedence constraints on the schedule variables, which, in turn, corresponds to a family \( A_{ij}^s \) of arcs in the graph of the routes \( R \).

The LD amounts to finding a minimum cost schedule \( t \) such that all pairs of trains only meet in stations. For every \( \{i,j\} \subseteq T \), and every \( s \in S \), we introduce a binary
variable $y_{ij}^s$ and we let $y_{ij}^s = 1$ if $i$ and $j$ meet in $s$, and 0 otherwise.

The LD can therefore be formulated as follows:

\[
\begin{align*}
\min & \quad c(t) \\
\text{s.t.} & \quad t_v - t_u \geq W_{uv}, \quad (u, v) \in A \\
& \quad t_v - t_u \geq M (y_{ij}^s - 1) + \epsilon, \quad (u, v) \in A_{ij}, s \in S, \{i, j\} \subseteq T \\
& \quad \sum_{s \in S} y_{ij}^s = 1, \quad \{i, j\} \subseteq T \\
& \quad t \in \mathbb{R}_+^V, \quad y \text{ binary}
\end{align*}
\] (5)

where $M$ is a large suitable constant. Also, since $c(t)$ is convex and piece-wise linear, it can be easily linearized by adding suitable variables and constraints, and (5) can be turned into a MILP.

Let $(\bar{t}, \bar{y})$ be a feasible solution to (5). Then the binary vector $\bar{y}$ is called a meeting.

We discuss now a property of meetings with crucial consequences on the solution algorithm. We recall here that an undirected graph $G = (V, E)$ is called an interval graph if it is the intersection graph of intervals of the real line, i.e. the nodes of $G$ correspond to intervals and there is an edge between two nodes if and only if the corresponding intervals overlap.

\textbf{Lemma 3.1} Let $\bar{y}$ be a meeting and let $\bar{y}_s \in \{0, 1\}^{(|T|)}$ be the subvector of $\bar{y}$ associated with station $s$. Then $\bar{y}_s$ is the incidence vector of the edges of an interval graph.

\textbf{Proof.} Since $\bar{y}$ is a meeting, there exists $\bar{t} \in \mathbb{R}_+^V$ such that $(\bar{t}, \bar{y})$ is feasible to (5). For a train $i \in T$, let $Q_i^s$ be the time interval (possibly empty) in which the train is in station $s$ according to the schedule $\bar{t}$. Let $G_s = (T, E_s)$ be the interval graph associated with the time intervals $\{Q_i^1, \ldots, Q_i^{|T|}\}$. Now, $\bar{y}_{ij}^s = 1$ if and only if $Q_i^s \cap Q_j^s \neq \emptyset$, that is if and only if $\{i, j\} \in E_s$. \hfill \qed

We denote by $G(y_s)$ the interval graph associated with the meeting $y$ and station $s$. It is trivial to see that, given an interval graph $H = (V, E)$ it is possible to build an instance of the LD with solution $(t, y)$ so that $G(y_s) = H$, i.e. $y_s$ is the incidence vector of the edges of $H$.

Now, recall that a clique in a graph is a subset of nodes all pairwise adjacent. By the Helly property we have the following simple result:

\textbf{Remark 3.2} Let $(t, y)$ be a solution to the LD. Let $K \subseteq T$ be a subset of trains and let $s \in S$ be a station. Then the trains in $K$ are simultaneously in station $s$ (according to the schedule $t$) if and only if $K$ is a clique of $G(y_s)$.

A solution $(t, y)$ to the LD (5) cannot in general be extended to a solution of the RTD. Indeed, it may be impossible to accommodate trains in a station according to
schedule \( t \) (which establishes when trains enter and leave the station). The corresponding feasibility problem is the SD earlier introduced and is discussed in the next section. We will also show how to extend (5) to represent such feasibility problem so as to obtain a MILP for the RTD. Any feasible solution \((t,y)\) to the latter will then be feasible also for all the SDs associated with the stations (namely, dispatching all stations of the railway).

We conclude this section by briefly discussing the (immediate) extension to the double-track line case. The only relevant difference is that crossing trains do not necessarily meet in stations, but they can also "meet" on a pair of parallel tracks\(^5\). Let \( B^D \subseteq B \) be the subset of double-tracks. We introduce a binary variable \( z_{ij}^b \) for every pair of crossing trains \( \{i,j\} \) and every double-track \( b \in B^D \), which is one if and only if \( i \) and \( j \) meet in \( b \). Then, for every pair of crossing trains \( \{i,j\} \), constraint (5.iii) is replaced by the following

\[
\sum_{s \in S} y_{sij} + \sum_{b \in B^D} z_{ij}^b = 1
\]

(6)

Also, if trains \( i \) and \( j \) meet on a double-track \( b \), then \( i \) enters \( b \) before \( j \) reaches the station following \( b \) on its route and viceversa. This can be expressed by a pair of precedence constraints or, equivalently, a pair of arcs \( A_{ij}^b \) on the graph of the routes.

Then, for every pair of crossing trains \( \{i,j\} \), the following constraints must be included in (5):

\[
t_v - t_u \geq M(z_{ij}^b - 1) + \epsilon, \quad (u,v) \in A_{ij}^b, b \in B^D
\]

(7)

4 Compact VS non-compact formulations for station dispatching

We focus now our attention on a station \( s \). A solution to the LD provides a timetable for \( s \), that is the time in which each train enters and leaves \( s \). In its more general version, the SD requires finding, for each train entering or leaving \( s \), a route in \( s \) and a schedule of the movements of the train along its route so that input and exit times from the station agree with a given timetable. This general SD closely resembles its off-line version (the \textit{Train Platforming problem}, see [11]). However, in most practical contexts and in particular in our specific setting, we can make reasonable assumptions that make the problem simpler.

First, in single-track lines and in particular in those considered in our test-bed, stations are usually small, like the one in Figure 1. Basically, for a given platform, there is only one route going through it (two, if you consider opposite directions). In other

\(^5\)In principle, a follower could catch up and pass the leading train on a double-track section (\textit{parallel run}). However, this manoeuvre is very ticklish and can only be engaged by human operators.
words, there is a one-to-one correspondence between platforms and routes for a given train, and if we choose a platform for train \( i \), then we also establish the station route for \( i \). A second assumption is that the running time from the entrance stopping point to any given platform is (approximatively) the same for all trains and all platforms. So, we do not add further delay to a train by selecting, say, platform \( b \) instead of platform \( a \).

Thanks to these two assumptions, the SD is reduced to deciding whether the platforms in a station suffice to accommodate all incoming trains, which, in turn, only depends on the meeting vector \( y \).

We state now more formally the (no routing) SD for a given station \( s \).

**Problem 4.1 (SD)** Let \( P \) be the set of platforms, let \( T \) be the set of controlled trains and let \( y_s \) be a feasible meeting in the station. For every train \( i \in T \) denote by \( P(i) \subseteq P \) the set of platforms that can accommodate train \( i \). Then the SD is the problem of assigning to each \( i \in T \) a platform in \( P(i) \) so that \( i \) and \( j \) receive a different platform whenever \( y_{ij}^s = 1 \).

Given a undirected graph \( G = (V, E) \), a coloring is a function \( c : V \to N \) such that \( c(i) \neq c(j) \) for all \( \{i, j\} \in E \). A \( k \)-coloring is a coloring such that \( c(i) \leq k \) for all \( i \in V \). Given sets \( L(i) \subseteq \{1, \ldots, k\} \) for \( i \in V \), a list coloring of \( G \) is a coloring \( c \) with \( c(i) \in L(i) \). Consider a function \( \mu : V \to N \). A \( \mu \)-coloring is a coloring \( c \) of \( G \) with \( c(i) \leq \mu(i) \) for every \( i \in V \). The coloring, \( k \)-coloring, list-coloring and \( \mu \)-coloring problems amount to establishing if a graph \( G \) admits a coloring, a \( k \)-coloring, a list coloring and a \( \mu \)-coloring, respectively. The following complexity results are surveyed in [5]: for interval graphs, the coloring problem and the \( k \)-coloring problems are easy, the list coloring and the \( \mu \)-coloring problems are NP-complete.

It is not difficult to see that the SD amounts to finding a list coloring of \( G(y_s) \), with \( L(i) = P(i) \) for every node \( i \). In the previous section we have seen that \( G(y_s) \) can actually be any interval graph. It immediately follows the next

**Theorem 4.2** The SD is NP-complete.

**Proof.** Reduction from list-coloring in interval graphs.

However, for most stations in a single-track line, a more treatable situation occurs, namely \( P(i) = P \) for all \( i \) and every train can be accommodated in any of the platforms 1, \ldots, \( k \) of the station. We call this case the all-good SD.

**Lemma 4.3** The all-good SD is easy.

**Proof.** When all color lists are equal to \( \{1, \ldots, k\} \), the list coloring problem reduces to the \( k \)-coloring problem. The \( k \)-coloring problem is easy for interval graphs.

---

\(^6\)Clearly, this does not hold for larger stations, where several routes go through the same platform.
The platforms of a station may be partitioned according to the incoming direction of trains, as often happens in double-track lines. Namely, trains coming from one direction can only access the platform in a class of the partition. It not difficult to see that the above result generalizes to the following:

**Corollary 4.4** Let \( T_1, \ldots, T_k \) be a partition of the trains \( T \) and let \( P_1, \ldots, P_k \) be a partition of the platforms \( P \). Assume that a train in \( T_q \) can access all platforms in \( P_q \), \( q = 1, \ldots, k \), and no other platforms. Then the corresponding SD is easy.

Since an interval graph admits a \( k \)-coloring if and only if it does not contain a clique of cardinality larger than \( k \), by Remark 3.2 we have the following

**Corollary 4.5** The all-good SD problem for station \( s \) has solution if and only if there are never more than \(|P|\) trains simultaneously in \( s \).

Observe that for the general SD, the above condition is not sufficient to ensure that a solution exists.

Finally, there is an intermediate case which occurs in practice. Namely, when platforms and trains have variable lengths and a train can only be accommodated on a platform which is at least as long. We call this the hierarchical SD. We have that:

**Lemma 4.6** The hierarchical SD is NP-complete.

**Proof.** Reduction from \( \mu \)-coloring on unit interval-graphs. A unit interval graph is the intersection graph of unit length intervals. Observe that every \( \mu \)-coloring uses at most \( k_\mu = \max_{i \in V} \mu(i) \) colors. So, given the function \( \mu \) and a unit interval graph \( H = (V, E) \) we construct an instance of the hierarchical SD in the following way. We consider a single station line. We let the set of trains \( T = V \), the platforms \( P = \{1, \ldots, k_\mu\} \) and the meeting \( y \) be the incidence vector of the edges of \( H \) (i.e. \( G(y) = H \)). Next, for each train \( i \in T \), we define its length as \( l_T(i) = M - \mu(i) \), where \( M \) is a large real number; similarly, for each platform \( p \in P \) we let its length be \( l_P(p) = M - p \). Suppose that the associated hierarchical SD is feasible, and let \( c : T \to P \) be an assignment of platforms to trains. Then \( c \) is also a \( \mu \)-coloring of \( H \). In fact, since \( c(i) \neq c(j) \) for all \( \{i, j\} \in E \), \( c \) is a coloring of \( H \). Also, for each \( i \in T \) we have \( l_P(c(i)) \geq l_T(i) \), which implies \( M - c(i) \geq M - \mu(i) \), which becomes \( c(i) \leq \mu(i) \), and \( c \) is a \( \mu \)-coloring of \( H \).

An alternative way to derive the above complexity results is to exploit the relation between the (no routing) SD and the Interval Scheduling problem (see [23]). Given a set of jobs each to be processed by one of a family of identical machines in a specified time interval, the basic Interval Scheduling problem amounts to establishing if the machines suffice to process all the jobs, provided that no two jobs are processed simultaneously on the same machine. One can show that the basic Interval Scheduling problem is easy
and is equivalent to $k$-coloring the intersection (interval) graph of the time intervals. It is not hard to see that the all-good SD is equivalent to this basic version of Interval Scheduling. Paper [23] also introduces the \textit{Hierarchical Interval Scheduling Problem} and shows that it is NP-complete. Without getting into details, it is possible to show that the latter is equivalent to the Hierarchical SD.

Once again we remark that for more complex stations, when for example multiple and conflicting routings to access or leave the same platforms exist, then more complex models should also apply, as for example in [40]. Nevertheless, the decomposition principle here introduced along with the master-slave solution approach are still exploitable. In addition, also in these cases valid Benders’ feasibility cuts can still be generated by solving suitable instances of the all-good SD, as we will show in Section 5. In the sequel of this section we show how this can be done effectively.

### 4.1 MILP models for the SD

In what follows we discuss two different approaches to the solution of the all-good SD. The first leads to a compact formulation. In contrast, the second may lead to a number of constraints which grows exponentially with the number of trains and of platforms. Remarkably, by exploiting the master/slave scheme naturally stemming from our decomposition, the non-compact approach has proven to be significantly more effective in practice, as we will show in Section 6. From here on, we shall refer to the number $c_s$ of platforms of station $s$ as \textit{station capacity}.

**A compact, flow based representation of the all-good SD.** Our purpose is to "embed" in (5) feasibility cuts from the SDs in in order to derive a MILP for the RTD. To this end, we will express the all-good SD in terms of a family of linear inequalities in variables $t$ and $y$, introducing new variables when necessary. We will do this by defining a suitable network flow problem which, in turn, can be modeled by linear programming.

Let $(t, y)$ be a solution to the LD, let $s \in S$ be a station and $i, j \in T$ be two distinct trains going through $s$. We say that $j$ is a successor of $i$ in $s$ (according to $(t, y)$) if $i$ leaves $s$ before $j$ enters $s$. We now introduce, for every ordered pair $(i, j)$ of distinct trains and every station $s \in \{1, \ldots, |S|\}$, the quantity $x_{ij}^s$ which is 1 if $j$ is a successor of $i$ in station $s$ and 0 otherwise. It is not difficult to see that $x$ can be obtained from $y$.

In fact, if $i$ runs from station 1 to station $|S|$ and $j$ from $|S|$ to 1 (so they run in opposite directions), and they meet in station $1 \leq k \leq |S|$ ($y_{ij}^k = 1$), then $i$ is a successor of $j$ in every station $s > k$ and $j$ follows $i$ in every station $s < k$. Assuming $i < j$, the above conditions can be expressed by the following constraints:

$$x_{si}^j = \sum_{q<s} y_{ij}^q, \quad s \in S$$

and

$$x_{sj}^i = 1 - \sum_{q<s} y_{ij}^q, \quad s \in S$$
Similar transformations may be derived for a pair of followers. In general, there exists an affine transformation from \( y \) to \( x \), i.e.

\[
x = Qy + q
\]

where \( Q \) and \( q \) are suitable matrices.

Now, we can interpret station platforms as (unitary) resources that can be supplied to trains. Then a train \( j \) receives a platform \( p \) either from a previous train \( i \) that used platform \( p \) or "directly" from station \( s \) (if no previous trains have used \( p \)). Following this interpretation, we can represent the SD as a network flow problem. Informally, station \( s \) can be represented by a supply node (it supplies up to \( c_s \) units of resource) and every train \( i \) can act both as a demand node and a supply node, since it can supply 1 unit of resource to successive trains.

We consider now a station \( s \) and a meeting vector \( \tilde{y} \), along with the corresponding successors vector \( \tilde{x} \). For sake of simplicity, we assume that every train in \( T \) goes through \( s \). We introduce the support graph \( \mathcal{N}(s, \tilde{x}) = (\{r, p\} \cup U \cup W, E) \), where \( U = \{u_1, \ldots, u_{|T|}\} \), \( W = \{w_1, \ldots, w_{|T|}\} \). Let the arc set \( E = E_r \cup E_U \cup E_W \cup E_p \cup \{(p, r)\} \), where \( E_r = \{(r, u_j) : j \in T\} \), \( E_U = \{(u_j, w_j) : j \in T\} \), \( E_W = \{(w_i, u_j) : i, j \in T, i \neq j\} \), \( E_p = \{(w_j, p) : j \in T\} \). With each arc \( e \in E \) we associate lower bound \( l_e \) and upper bound \( f_e \). In particular, \( l_e = 1 \) for \( e \in E_U \) and \( l_e = 0 \) for \( e \in E \setminus E_U \). Also, \( f_e = 1 \) for \( e \in E_r \cup E_U \cup E_p \), \( f_{(w_i, u_j)} = \tilde{x}_{ij}^s \) for \( (w_i, u_j) \in E_W \) and \( f_{pr} = c_s \). A representation of a generic support graph is given in Figure 7.

![Figure 7: The support graph. Lower and upper bounds are shown between brackets for some representative arcs](image)

We have the following

**Theorem 4.7** The all-good SD has a solution if and only if the graph \( \mathcal{N}(s, \tilde{x}) \) has a circulation satisfying all lower and upper bounds.

We give the sufficiency proof of this theorem in the Appendix. The necessity (constructive) proof is simpler and is omitted.
Incidentally, it can be easily shown that our network flow problem actually solves the (equivalent) problem of coloring an interval graph with $c_s$ colors. There exist alternative representations of the $k$-coloring problem for interval graphs as network flows, like the one presented by Carlisle and Lloyd in [12]. However, we were not able to find a suitable extension of (5) to represent the problem described in [12] and we developed a different approach.

Our circulation problems can be readily expressed as linear programs ([1]) in the $x$ variables (plus standard flow variables). By using transformation (8) we couple the circulation problems to (5) so as to obtain a MILP for the RTD. However, as we will show in the computational section, the approach discussed next has proven to be more effective in solving the instances of the RTD problem in our test-bed.

A non-compact formulation. Consider a station $s \in S$ with $c_s$ platforms and let $(t, y)$ be a solution to the LD. Then we can assign the $c_s$ platforms of $s$ to incoming trains if and only if the (interval) graph $G(y_s)$ can be colored with $c_s$ colors. In turn, this can be done if and only if $G(y_s)$ does not contain a clique of cardinality strictly larger than $c_s$ (see, for example, [35]). Any such clique will in turn contain a clique $K$ of size $c_s + 1$. The number of edges of $K$ is exactly $\binom{c_s + 1}{2}$, or, equivalently, $\sum_{\{i,j\} \subseteq K} y_{ij} = \frac{1}{2}(c_s + 1)c_s$. In other words, the meeting $y$ does not violate station capacity if and only if, for all $s \in S$, we have:

$$\sum_{\{i,j\} \subseteq K} y_{ij} \leq \frac{1}{2}(c_s + 1)c_s - 1$$

for all $K \subseteq T$ with $|K| = c_s + 1$.

5 Solution Algorithm

We are finally able to formulate the RTD for the single- and double-track case as a Mixed Integer Linear Program by coupling constraints (9) and program (5) and linearizing the objective function:
\[
\begin{align*}
\min & \quad c(t) \\
\text{s.t.} & \quad (i) \quad t_v - t_u \geq W_{uv}, \quad (u, v) \in A \\
& \quad (ii) \quad t_v - t_u \geq M(y_{ij} - 1), \quad (u, v) \in A_{ik}^j, k \in S, \{i, j\} \subseteq T \\
& \quad (iii) \quad t_v - t_u \geq M(z_{ij} - 1), \quad (u, v) \in A_{ib}^j, b \in B^D, \{i, j\} \subseteq T, i, j \text{ crossing} \\
& \quad (iv) \quad \sum_{s \in S} y_{ij}^s + \sum_{b \in B^D} z_{ij}^b = 1, \quad \{i, j\} \subseteq T \\
& \quad (v) \quad \sum_{\{i, j\} \subseteq K} y_{ij}^s \leq \frac{1}{2}(c_s + 1)c_s - 1, \quad s \in S, K \subseteq T, |K| = c_s + 1 \\
t \in \mathbb{R}_+^V, \quad y, z \text{ binary}
\end{align*}
\] (10)

To simplify the notation, we write constraint (10.iv) in the same form for every pair of trains, by assuming \( z_{ij}^b = 0 \) for all pairs of followers \( i, j \) and all \( b \in B^D \). The alternative compact formulation is obtained by replacing constraints (10.v) with the inequalities defined in the circulation problem on the network \( N(s, x) \) for all \( s \), plus the affine transformation (8) from \( x \) to \( y \).

One major drawback of the non-compact formulation is that the number of constraints (10.v) can become exponentially large with respect to the number of trains and capacity of the stations. Also, the number of constraints (10.ii) can become very large in practice, even in our instances with a relatively small number of trains. For this reason we resort to the delayed row generation approach ([2]) which we summarize next. We start by selecting an initial subset of constraints. Then, in each node of the branching tree, we (i) solve the current linear relaxation (ii) check if the current fractional solution violates any of the neglected constraints (separation) (iii) add the violated constraints to the current program and iterate. Following this scheme, our initial formulation contains only (all) constraints (10.i).

We first focus on the generation of constraints (10.v) deriving from the decomposition of our original problem. In the classical Benders’ decomposition algorithm (see, e.g., [30]), a relaxed problem is solved in every node of the branching tree and Benders’ cuts violated by the current fractional solution are generated. In contrast, in their combinatorial variant, Codato and Fischetti ([14]) prefer to solve to (integral) optimality the original master problem; then, violated combinatorial Benders’ cuts are generated and added, and the revised master problem is again solved to integral optimality. We follow a somehow intermediate path by generating violated constraints during the branching process. However, rather than generating constraints of type (10.v) in every node of the branching tree, we limit the generation to the nodes corresponding to integer solutions. In this way, the slave problem is precisely the all-good SD described in Section 4 and the separation is easy. Indeed, when \( y \) is binary (and no other constraints are violated), the graph \( G(y_s) \) is interval (Lemma 3.1) for each \( s \in S \). Then, finding an inequality of type (10.v) violated by \( y \) amounts to finding, for each \( s \in S \), a maximum cardinality
clique in the interval graph $G(y_s)$, which in turn can be done in $O(|T| \log |T|)$ time (see [19]). Furthermore, the algorithm does not need to solve several integer problems to optimality as in [14].

An open question is the complexity of separating (10.v) for fractional solutions. Actually, if $y$ can assume any fractional value, then the separation problem for (10.v) reduces to the Maximum Edge-Weighted Clique problem in undirected graphs. The latter is known to be an NP-hard problem (see [25]), leaving very little hope to solve fractional separation efficiently.

Concerning inequalities (10.ii) and (10.iii), they are also only separated (by inspection) in the integer nodes of the branching tree.

Once a feasible meeting $y$ is found, it is easy to obtain a platform assignment by coloring the interval graphs $G(y_s)$ for all $s \in S$. When the hypothesis of the all-good SD do not hold, then we need to resort to different approaches. However, observe that constraints (10.v) remain valid even when more complicated station models apply, but they do not suffice to provide a formulation. Notably, one can show that for all the variants of the SD suitable cuts in the $y$ variables still suffice to represent infeasibility.

An initial feasible solution is also provided to the algorithm by running the heuristic procedure described below.

A final interesting remark is that our decomposition and row generation approach mimics, in some sense, the actual behavior of human dispatchers. A violated constraint (10.ii) or (10.iii) corresponds to a so called line conflict, that is a situation in which two trains will (if no recourse action is taken) occupy incompatible track sections at the same time. Line conflicts are detected by dispatchers and prevented by establishing a correct meeting station for the conflicting trains. Dispatchers then induce drivers to follow their decisions by switching suitable traffic signals to red light. Adding a constraint of type (10.ii) is the mathematical equivalent of activating a red signal.

Our exact decomposition algorithm has been embedded both in a system developed by Bombardier Transportation in Italy and in a system developed by SINTEF ([36]) in Norway for the railway operator (JVB[22]) and train companies (NSB[29], Flytoget[18], CargoNet[13]). The latter has been in operation on a line in Norway since February 2014, fully exploiting the exact approach presented in this paper. The former has been in operation since 2011 but the use of the exact algorithm has not yet been validated due to the operative rules currently in place. In order to comply with such rules, an ad-hoc heuristic was developed and deployed. In the following paragraph we give a brief description of how it works.

A heuristic for Train Dispatching. The algorithm briefly described next is operating on several railway lines in Italy (see Section 6). The heuristic essentially extends the current decision process of dispatchers with two major enhancements: 1. evaluates a considerably larger number of alternatives 2. provides complete solutions in very short time (whereas dispatchers solve one conflict at a time).

---

7 An on-field test campaign has been pending since 2013
Again, we decompose the problem into LD and SD; however, the sub-problems in this case are solved heuristically, following RFI's ([32]) prioritization rules. In particular, at each iteration, potentially conflicting pairs of trains are identified and conflicts are ordered chronologically. The first conflict is then "solved" by establishing, for the corresponding pair of trains, a meeting point, station or double-track. Any choice causes a delay for one of the two trains, which is forced to wait for the other. Possible meeting points are then ordered increasingly with respect to such waiting times and visited accordingly. If the current meeting point is a station, then the trains are assigned to platforms and station routes optimally without violating RFI’s rules. In some cases, solutions satisfying all rules may not exist, in other, the capacity of the station would be violated. Such infeasible choices are consequently disregarded and the following meeting point in the ordering is examined. Once the conflict is solved, corresponding precedence constraints are added to the problem and the process iterates. In principle, this algorithm may fail to find a feasible solution: however, this seldom happens in practice. When this occurs, alternative dispatching decisions will be taken directly by dispatchers in charge.

6 Computational Results

The target of our computational tests was twofold. Firstly, we wanted to identify the most effective approach to solve the RTD between the decomposition and compact formulations. Secondly, we confronted the best approach (the non-compact formulation) with the current practice. Finally, in Section 7 we will describe a real-life operative implementation of an exact algorithm based on this decomposition approach.

We initially ran our tests on a number of real-life instances of single- and double-track railways in Italy, in regions with considerably different topography and network status quo. In particular, we focused our experiments on three lines: Trento - Bassano del Grappa, Foligno - Orte and Foligno - Falconara. Details about these lines are given in Table 1. All instances from these lines were provided by Bombardier Transportation, extracted at peak hours, and refer to existing trains actually running on the above lines.

<table>
<thead>
<tr>
<th>Line</th>
<th>Abbr.</th>
<th>Stops</th>
<th>Stations</th>
<th>Length (m)</th>
<th>S.T.</th>
<th>D.T.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trento - Bassano</td>
<td>T-BG</td>
<td>22</td>
<td>14</td>
<td>95711</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Foligno - Orte</td>
<td>F-O</td>
<td>13</td>
<td>10</td>
<td>82018</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Falconara - Foligno</td>
<td>F-F</td>
<td>24</td>
<td>17</td>
<td>119612</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 1: Infrastructure details. S.T. stands for Single-Track, D.T. stands for Double-Track

A 60 second time-limit was fixed for our tests, which is regarded as an acceptable time span for a dispatcher’s decision process on these lines. In practice, real life require-
ments may be less stringent. As for the convex, piece-wise linear objective function described in Section 2, for each train \( i \) and each station \( s \) we fixed 3 breakpoints on the time axis (the x-axis), to the right of the aimed arrival time \( W^i_s \), at \( W^i_s + 1min \), \( W^i_s + 3min \) and \( W^i_s + 6min \), respectively. The aimed arrival time and three breakpoints identify 3 segments and a halfline with slopes, 0, 1, 3 and 9, respectively. Also, all time durations \( W \) in constraints (10.i) are expressed as an integer number of seconds.

**Implementation details.** Tests on instances in Tables 2, 3 were run using a Dell - PowerEdge® M910 with a 64 Bit Windows Server 2008 R2 Enterprise SP1 OS, 4 Intel® Xeon L7555 @ 1,86GHz CPUs, a 128GB RAM, and CPLEX® 12.3 as a solver. All other tests were run on an Intel® Core(tm) i7-2640M CPU 870 2.80GHz machine using CPLEX® 12.2.

### 6.1 Confronting compact formulation and decomposition

This set of experiments was designed to determine the best approach between the compact, flow-based formulation and the non compact, decomposition based one. In order to obtain fair comparisons we chose not to provide initial upper bounds. It emerges quite clearly that the compact formulation proved to be less effective than the non compact one. An overview of computed results is shown in Tables 2, 3, where \( C \) stands for Compact, \( NC \) stands for Non-Compact. In particular, in Table 2 we show results for (representative) instances for which the algorithm was able to find the optimal solution within the time limits, which was the most common outcome. However, for a few instances, the algorithm was not able to prove optimality within time limits (some examples are shown in Table 3).

<table>
<thead>
<tr>
<th>ID</th>
<th>Line</th>
<th>#Trains</th>
<th>Initial Rows</th>
<th>Generated</th>
<th>Computation Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>NC</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>T-BG</td>
<td>29</td>
<td>9459</td>
<td>1333</td>
<td>212</td>
</tr>
<tr>
<td>2</td>
<td>T-BG</td>
<td>28</td>
<td>8980</td>
<td>1302</td>
<td>246</td>
</tr>
<tr>
<td>3</td>
<td>T-BG</td>
<td>29</td>
<td>9691</td>
<td>1351</td>
<td>124</td>
</tr>
<tr>
<td>4</td>
<td>T-BG</td>
<td>29</td>
<td>4794</td>
<td>912</td>
<td>762</td>
</tr>
<tr>
<td>5</td>
<td>T-BG</td>
<td>22</td>
<td>4858</td>
<td>918</td>
<td>688</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2:** Computational results: instances solved to optimality within time limits. Column "Generated" refers to the number of rows generated during the branch-and-cut process.

---

8This much depends on distances between stations. Norwegian dispatchers allow, for instance, up to 10 minutes reaction time for some single-track lines [33]
In our experiments, on average the non-compact formulation outperformed the compact, flow based one, both in terms of solution quality and computation time. In most cases, the algorithm(s) found optimal solutions within a few seconds, an acceptable time for dispatchers. In other cases, the process did not terminate with optimal solutions within the time limit.

In Table 3 we show the algorithm’s performance, for both formulations, for five of such instances from T-BG. For this experiment the time limit was raised to 300 seconds. We report the number of controlled trains (column "Trains"), the best solution values and gap values found for increasing time. In each experiment, gap is computed in a standard fashion, by comparing the current best solution value (UB) with the value (LB) of the best linear relaxation so far, that is $(UB - LB)/LB$. Both for the compact and for the non-compact formulation, gap can (only) decrease with time, as more cuts are added or a better incumbent solution can be found. In most cases, the non-compact formulation produced better solutions and terminated with a gap which was at most 52%. For one instance ($i_3$) C terminated with a better solution, while in another ($i_2$) C found a solution with lower gap value after 30 and 60 seconds (although NC produced the best solution at the time limit). However, in the last instance ($i_5$), C was not able to produce a feasible solution at all.

<table>
<thead>
<tr>
<th>ID</th>
<th>F</th>
<th>Trains</th>
<th>10 s gap</th>
<th>sol</th>
<th>30 s gap</th>
<th>sol</th>
<th>60 s gap</th>
<th>sol</th>
<th>180 s gap</th>
<th>sol</th>
<th>300 s gap</th>
<th>sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>C</td>
<td>19</td>
<td>93%</td>
<td>1575</td>
<td>93%</td>
<td>1575</td>
<td>80%</td>
<td>1575</td>
<td>70%</td>
<td>1575</td>
<td>69%</td>
<td>1575</td>
</tr>
<tr>
<td>i1</td>
<td>NC</td>
<td>19</td>
<td>0%</td>
<td>1219</td>
<td>0%</td>
<td>1219</td>
<td>0%</td>
<td>1219</td>
<td>0%</td>
<td>1219</td>
<td>0%</td>
<td>1219</td>
</tr>
<tr>
<td>i2</td>
<td>C</td>
<td>28</td>
<td>-</td>
<td>-</td>
<td>48%</td>
<td>1799</td>
<td>48%</td>
<td>1799</td>
<td>48%</td>
<td>1799</td>
<td>48%</td>
<td>1799</td>
</tr>
<tr>
<td>i2</td>
<td>NC</td>
<td>28</td>
<td>57%</td>
<td>2200</td>
<td>57%</td>
<td>2200</td>
<td>57%</td>
<td>2200</td>
<td>22%</td>
<td>1262</td>
<td>18%</td>
<td>1259</td>
</tr>
<tr>
<td>i3</td>
<td>C</td>
<td>28</td>
<td>-</td>
<td>-</td>
<td>24%</td>
<td>1266</td>
<td>24%</td>
<td>1266</td>
<td>20%</td>
<td>1266</td>
<td>15%</td>
<td>1266</td>
</tr>
<tr>
<td>i3</td>
<td>NC</td>
<td>28</td>
<td>53%</td>
<td>2153</td>
<td>43%</td>
<td>1766</td>
<td>35%</td>
<td>1546</td>
<td>33%</td>
<td>1546</td>
<td>33%</td>
<td>1546</td>
</tr>
<tr>
<td>i4</td>
<td>C</td>
<td>28</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>56%</td>
<td>2765</td>
<td>56%</td>
<td>2765</td>
<td>51%</td>
<td>2765</td>
</tr>
<tr>
<td>i4</td>
<td>NC</td>
<td>28</td>
<td>36%</td>
<td>1922</td>
<td>34%</td>
<td>1922</td>
<td>33%</td>
<td>1922</td>
<td>33%</td>
<td>1922</td>
<td>32%</td>
<td>1922</td>
</tr>
<tr>
<td>i5</td>
<td>C</td>
<td>28</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>i5</td>
<td>NC</td>
<td>28</td>
<td>54%</td>
<td>3307</td>
<td>53%</td>
<td>3307</td>
<td>53%</td>
<td>3307</td>
<td>52%</td>
<td>3307</td>
<td>52%</td>
<td>3307</td>
</tr>
</tbody>
</table>

Table 3: Instances where the algorithm could not prove optimality within time limits (300 seconds) for both formulations.

6.2 Evaluating the decomposition approach

These tests showed that on average algorithm $NC$ outperforms algorithm $C$ both in terms of computation time and quality of solutions. Next, we evaluate $NC$ on more complex lines, with both single and double-tracks, and a higher number of controlled
trains. As the heuristic algorithm described in Section 5 is already in operation on these lines, we are able to confront the solutions generated by the new exact approach with the decisions currently being carried out by dispatchers on these lines. Noticeably, statistics show that dispatchers follow the decisions taken by the heuristic algorithm more than 90% of times. Furthermore, in the remaining cases, it is not possible to determine from the available data the actual reasons for such discrepancy, which may be caused by corrupted input data. In Table 4, we present a summary of computed results for 10 representative instances from the $F-O$ line. Note that E stands for Exact and H stands for Heuristic and that, again, the time limit was fixed to 60 seconds. In Column $Gap$ we confront the best solution found value ($UB$) with the best lower bound ($LB$) returned by CPLEX. In particular, the gap is again computed as $\frac{UB-LB}{LB}$.

<table>
<thead>
<tr>
<th># Trains</th>
<th>Solution</th>
<th>Optimal</th>
<th>Gap</th>
<th># Conflicts</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>H</td>
<td>E</td>
<td>H</td>
<td>E</td>
<td>H</td>
</tr>
<tr>
<td>45</td>
<td>696</td>
<td>1010</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>46</td>
<td>494</td>
<td>1397</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>48</td>
<td>203</td>
<td>293</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>50</td>
<td>277</td>
<td>314</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>49</td>
<td>360</td>
<td>422</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>50</td>
<td>263</td>
<td>346</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>50</td>
<td>503</td>
<td>653</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>50</td>
<td>691</td>
<td>759</td>
<td>yes</td>
<td>no</td>
<td>0%</td>
</tr>
<tr>
<td>50</td>
<td>180</td>
<td>210</td>
<td>no</td>
<td>no</td>
<td>13%</td>
</tr>
<tr>
<td>50</td>
<td>690</td>
<td>690</td>
<td>no</td>
<td>no</td>
<td>15%</td>
</tr>
</tbody>
</table>

Table 4: Computational results for the Foligno - Orte line (single/double-track), over an average 13 hour horizon. Note that, as heuristic computation times are always under a second, they have been omitted.

Also, in column "Conflicts" we indicate how many conflicts are solved and how many violations are found, for the heuristic and exact algorithm, respectively. Such indicator gives a (first) measure of how hard a given instance is (although other aspects may also have an impact). The reasoning behind this is the following: at time zero, a positive number of conflicts exists (primary conflicts) or the instance is trivial. As pointed out in Section 5 in describing the heuristic, solving one such conflict implies finding a feasible meet-pass decision for a given pair of trains. Imposing such decision will modify the real-time schedule and, possibly, generate other conflicts. This is exactly what happens in a purely sequential algorithm such as our heuristic or, indeed, when a dispatcher takes decisions on the line. In the exact algorithm, we apply delayed row generation, so conflicts take a slightly different but comparable meaning. In particular, a conflict in this case identifies the event of generating variables and constraints required to avoid a given pair of trains actually creating any conflict. Therefore, in general, a higher
number of conflicts will identify a harder instance. This trend can be observed in Table 4 (with the exception of some outliers). For the instances in such table, the highest number of trains on-line simultaneously was 8.

In most cases, the exact algorithm found optimal solutions within the time limit. In other cases, good feasible solutions were found. It is interesting to notice that, even when optimality is not proven, the quality of such solutions is generally higher than that of the corresponding heuristic ones.

To highlight the impact of dispatching decisions on the real-time timetable, in Table 5 we introduce a different performance indicator, namely train punctuality (i.e. distribution of delayed trains). Indeed, this is a powerful measure, which is immediately understood both by railway practitioners and general public. In Table 5, we report the average distribution of delayed trains for the exact and heuristic algorithms, computed by solving 500 instances, taken during a weekday in November 2012, for each of the single and double track lines considered in these experiments. Based on feedback from the railway operators, possible delays were subdivided in three macroscopic ranges: on time (less than 3 minutes), moderate delay between 3 and 6 minutes and delay greater than 6 minutes. Trains were then clustered according to the difference between expected and actual arrival time at destination.

<table>
<thead>
<tr>
<th>Line</th>
<th># Trains</th>
<th>Horizon (hrs)</th>
<th>On Time</th>
<th>Late between 3 and 6 mins</th>
<th>Later than 6 mins</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>E</td>
<td>H</td>
</tr>
<tr>
<td>T-BG</td>
<td>29</td>
<td>9</td>
<td>49%</td>
<td>75%</td>
<td>12%</td>
</tr>
<tr>
<td>F-O</td>
<td>40</td>
<td>12</td>
<td>83%</td>
<td>92%</td>
<td>9%</td>
</tr>
<tr>
<td>F-F</td>
<td>54</td>
<td>12</td>
<td>77%</td>
<td>81%</td>
<td>9%</td>
</tr>
</tbody>
</table>

**Table 5:** Punctuality distribution for 500 instances. Average figures.

As emerges from Table 5, in all cases, by applying the new approach the percentage of trains on time increased tangibly with respect to the current practice. The benefits of the exact algorithm were very evident for the Trento-Bassano line, with an increase in the number of trains on time of as much as 26%. For the two more complex lines, the increase was 4% for the Falconara-Foligno line and 9% for the Foligno-Orte line. Also, Table 5 shows how the average improvement in punctuality is not only due to slightly delayed trains arriving on time, but also to a decrease in the number of trains running severely late.

Overall, our results show how the implementation of the exact approach can significantly increase the quality of the real-time plan with respect to the current practice. In the next section, we describe such an implementation.
7 An operative implementation on the Stavanger-Moi line (Jærbane) in Norway

While our research was motivated by Italian applications (supported by Bombardier Transportation), the first operative system based on the ideas presented in this article was implemented in Norway in February 2014, backed by the network operator (JBV[22]) and train operating companies (NSB[29], Flytoget[18], CargoNet[13]). Such system was developed at SINTEF([36]) for the purpose of providing decision support to Norwegian dispatchers. After a first, very positive test campaign in the Trondheim area\(^9\), the main line in the Stavanger region (the \textit{Jærbane}\(^{10}\)) was agreed by all stakeholders to be a suitable candidate for the first real-life implementation in a Norwegian dispatching central. Figure 8 is a graphical representation of the Jærbane taken from the 2013 OpenTrack infrastructure model.

The Jærbane (123 km, 16 stations) is one of the few lines in Norway with both single- and double-tracks and, after the greater Oslo area, is the most trafficked line in Norway. This is due mostly to the number of local passenger trains which connect Stavanger and Sandnes, which amounts to around 40% of the line’s entire traffic. Figure 9 shows the Jærbane’s graphical timetable (train graph) for a weekday morning (06-12), where time and space are shown on the horizontal and vertical axis, respectively.

We present a brief description of the current setting: first of all, a server continuously acquires real-time data from the Traffic Management System. Each time an event occurs on the line (train reaches signaling point, delays are registered etc) the algorithm is run to identify a solution to the current RTD, which is then displayed on a screen in the Stavanger control center. Figures in Table 6 show that, in most cases, the solution found is optimal. Furthermore, we have noticed that, in the cases when this does not occur, corrupt real-time data in input is generally identified as main cause\(^{11}\).

Dispatchers interact with our system by confirming or modifying solutions and updating parameter settings such as slowdowns, interruptions, delays, cancellations, etc. Unfortunately, no historical data prior to February 2014 is available for the Jærbane. Consequently, direct comparisons with the dispatching process before such date (i.e. when dispatching was not supported by the currently operative system) are unattainable. However, we remark that comparisons with the current practice in Italy in Section 6 show how, even on moderately trafficked lines, using the exact methodology presented in this paper can significantly improve the current practice. Furthermore, dispatchers in Stavanger have clearly stated how the use of such system improves their work process and, in general, confirmed the quality of real-time solutions they are presented with.

In Figure 10 we give a measure of the traffic on the Jærbane on a weekday in March

\(^9\)Namely, the Dovrebane, line which connects Trondheim to Dombås, northern gateway to Oslo

\(^{10}\)Actually, the Jærbane technically comprises the region between Stavanger and Egersund, while the system also controls the part of the network which extends past Egersund to Moi

\(^{11}\)JBV is coming out with a tender for the renovation of Norway’s entire signalling system
Figure 8: The Jærbane (OpenTrack® infrastructure model 2013)

2014. Average information regarding number of controlled trains, trains simultaneously on the line and trains running late (i.e. delay larger than 3 minutes) is shown for different hours of the day. In Table 6 we present figures regarding the actual runs of the algorithm in March 2014, which show how most instances are solved to proven optimality within the time limit ("% Optimal"). Figures are presented for different days and include the number of runs considered\(^\text{12}\), the mean number of controlled trains ("# Trains"), mean and standard deviation of computation times ("Time", expressed in seconds). Last column ("Objective") indicates average objective value of the identified solutions. Optimization time horizon is set to 12 hours as agreed with the operator and train companies. Time limit for each run was set to 60 seconds by default.

On a side note, applying the ideas described in this paper to the dispatching system first operated in Stavanger has necessarily required taking in account further, specific albeit less significant details. Furthermore, algorithmic adaptations and speed-ups were

\(^{12}\)Runs are triggered when events occur on the line
necessary to comply with this application’s requirements. However, such aspects have been omitted in the paper for sake of clarity and brevity, as these authors believe they do not further assist the reader in understanding the essential aspects of the modelling approach and the relevance and impact of the application.

Due to the dispatchers’ positive feedback, the railway operator and train companies have confirmed their intention to extend such system to other dispatching centers in Norway.

8 Acknowledgments

We thank the anonymous referees and the editors for their precious comments. We also wish to thank for their contributions and assistance: Alessandro Mascis and Franco Pietrini (Bombardier Transportation), Paolo Perticaroli and Mauro Piacentini (Op-
<table>
<thead>
<tr>
<th>Day</th>
<th># Runs</th>
<th>% Optimal</th>
<th># Trains</th>
<th>Time(s) Mean</th>
<th>Std Dev</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>3588</td>
<td>85.5%</td>
<td>70</td>
<td>6.54</td>
<td>7.59</td>
<td>2511.27</td>
</tr>
<tr>
<td>27</td>
<td>3659</td>
<td>79.0%</td>
<td>76</td>
<td>10.73</td>
<td>12.29</td>
<td>6453.03</td>
</tr>
<tr>
<td>28</td>
<td>4986</td>
<td>89.1%</td>
<td>63</td>
<td>3.41</td>
<td>3.30</td>
<td>1259.09</td>
</tr>
<tr>
<td>29</td>
<td>3461</td>
<td>99.6%</td>
<td>30</td>
<td>0.45</td>
<td>0.29</td>
<td>1117.72</td>
</tr>
<tr>
<td>30</td>
<td>2859</td>
<td>99.9%</td>
<td>34</td>
<td>1.25</td>
<td>1.23</td>
<td>1255.72</td>
</tr>
</tbody>
</table>

Table 6: Figures for system performance on 5 days in March 2014.

tRail), Arnt-Gunnar Liium (SINTEF Applied Economics), Øyvind Bernhardt-Melin, Erik Natvig, Tone Norløff and Øyvind Stenersen (J BV), Øystein Risan (NSB).

References


[21] Indian Railway *Year Book 2010-11*, pp.22

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


9 Appendix

Sufficiency proof of Theorem 4.7 (the all-good SD in station $s$ and meeting $\bar{y}$ has a solution if $N(s, \bar{y})$ has a circulation).

**Proof.**

By contradiction, we assume that $N(s, \bar{x})$ has a circulation but the all-good SD has no solution. We use Hoffman’s circulation theorem, which states that $N$ does not have a circulation if and only if there exists a set of nodes $H$ such that $\sum_{e \in \delta^-(H)} l_e > \sum_{e \in \delta^+(H)} f_e$. So, assume that a platform assignment does not exist, then there exist $c_s + 1$ trains, say $Q = \{k, \ldots, k + c_s\} \subseteq T$, which are simultaneously in station $s$. We construct a cut by letting $H = \{p\} \cup \{w_j : j = \{k, \ldots, |T|\}\} \cup \{u_j : j = \{k + c_s, \ldots, |T|\}\}$.

We then have $\sum_{e \in \delta^-(H)} l_e = |Q| = c_s + 1$ since the only arcs with positive lower bound (the arcs in $E_U$) entering $H$ are precisely the arcs $(u_k, w_k), \ldots, (u_{k+c_s}, w_{k+c_s})$ (all other arcs in $E_U$ are either completely contained in $H$, for $j > k + c_s$, or in the complement $\bar{H}$ of $H$, for $j < k$). On the other hand, it is easy to see that the only arc with positive upper bound outgoing from $H$ is $(p, r)$, which implies $\sum_{e \in \delta^+(H)} f_e = c_s < c_s + 1 = \sum_{e \in \delta^-(H)} l_e$. In fact:

1. $E_r \cap \delta^+(H) = \emptyset$. Indeed, all arcs in $E_r$ are outgoing from $r$ and $r \in \bar{H}$.

2. $E_U \cap \delta^+(H) = \emptyset$. Indeed, $u_j \in H$ for $j = k + c_s, \ldots, |T|$. But then also $w_j \in H$ for $j = k + c_s, \ldots, |T|$.

3. $E_W \cap \delta^+(H) = \emptyset$. We must show that $(w_i, u_j) \notin E_W$ for $i \geq k$ and $j \leq k + c_s$. That is, we show that for $j \in \{1, \ldots, k, \ldots, k + c_s\}$ and $i \geq k$, then $j \notin Su(i)$. This is trivial for $i > k + c_s$ since $j \notin Su(i)$ for all $i > j$. Also, by assumption,
the trains in $Q = \{k, \ldots, k + c_s\}$ are simultaneously in the station, which implies that $j \notin Su(i)$ for all $j, i \in Q$.

4. $E_p \cap \delta^+(H) = \{(p, r)\}$. Trivial, since $p \in H$ and all arcs in $E_p \setminus \{(p, r)\}$ are incoming in $p$.

$\square$

34