

Report

An exact decomposition approach for the real-time Train Dispatching problem (v.2)

Follows SINTEF Technical Report A23274

Author(s)

Leonardo Cameron Lamorgese

Carlo Mannino

SINTEF IKT
SINTEF ICT

Address:
Postboks 124 Blindern
NO-0314 Oslo
NORWAY

Telephone: +47 73593000
Telefax: +47 22067350

postmottak.ikt@sintef.no
www.sintef.no
Enterprise /VAT No:
NO 948 007 029 MVA

Report

An exact decomposition approach for the real-time Train Dispatching problem (v.2)

Follows SINTEF Technical Report A23274

KEYWORDS:

Keywords

VERSION

v.2

DATE

2013-05-07

AUTHOR(S)

Leonardo Cameron Lamorgese
Carlo Mannino

CLIENT(S)

Jernbaneverket

CLIENT'S REF.

Tone Norløff

PROJECT NO.

102001009

NUMBER OF PAGES:

35

ABSTRACT

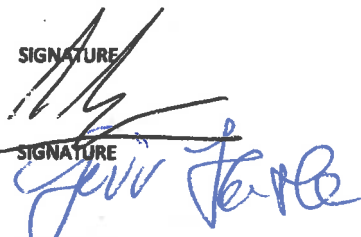
Trains movements on a railway network are regulated by official timetables. Deviations and delays occur quite often in practice, demanding fast re-scheduling and re-routing decisions in order to avoid conflicts and minimize overall delay. This is the real-time train dispatching problem. In contrast with the classic "holistic" approach, we show how to decompose the problem into smaller subproblems associated with the line and the stations. The 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 subproblems are modelled as mixed integer linear programs, with their specific sets of variables and constraints. Similarly to the classical Bender's decomposition approach, the slave and the master communicate through suitable feasibility cuts in the variables of the master. By applying our approach to a number of real-life instances from single and double-track lines in Italy, we were able to (quickly) find optimal or near-optimal solutions, with impressive improvements over the performances of the current operating control systems.

The new approach will be put in operation in such lines for an extensive on-field test-campaign as of April 2013.

PREPARED BY

Leonardo Lamorgese

SIGNATURE



CHECKED BY

Geir Hasle

SIGNATURE

APPROVED BY

Arnt-Gunnar Lium

SIGNATURE



REPORT NO.
SINTEF A24355

ISBN
978-82-14-05323-4

CLASSIFICATION
Unrestricted

CLASSIFICATION THIS PAGE
Unrestricted

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 subproblems associated with the line and the stations. The 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 subproblems are modeled as mixed integer linear programs, with their specific sets of variables and constraints. Similarly to the classical Bender's decomposition approach, the slave and the master communicate through suitable feasibility cuts in the variables of the master. By applying our approach to a number of real-life instances from single and double-track lines in Italy, we were able to (quickly) find optimal or near-optimal solutions, with impressive improvements over the performances of the current operating control systems. The new approach will be put in operation in such lines for an extensive on-field test-campaign as of April 2013.

1 Introduction

In a first and 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

*SINTEF ICT, Oslo, e-mail: leonardo.lamorgese@sintef.no, carlo.mannino@sintef.no

second phase, called *train platforming* or *track allocation* (see [7], [10]) 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 the dispatchers are actually doing (without being aware of it) is solving an optimization problem - and of a very tough nature. Following [34], we call this problem the *real-time Train Dispatching* problem (RTD).

In short, the RTD problem 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 problem 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* [16] 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 optimization can be found in [7], [8], [9], [10], [19], [34], [39]: actually the literature is much wider, and we refer to [25] for a survey. 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 problem 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 [28]. 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 [19]).

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

¹The problem of designing optimal production plans is of crucial relevance for railway operators. As pointed out in [25] *optimum resource allocation can make a difference between profit and loss for a railway transport company*

satisfied by any feasible schedule. A *disjunctive graph* ([3]), where disjunctions are represented 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 [29]. 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 [13] and [36]. 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 RTD problems. 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 [28], which handles small-scale metro instances, and [36], 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 problem: 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 [19]. In the same paper the author conjectures that the application of optimization to regular dispatching activities is imminent: in this paper we somehow confirm his conjecture.

Indeed, we introduce a new modeling approach to the RTD and a solution methodology which allow to overcome some of the limitations of natural big-M formulations and solve to optimality (or nearly) a number of real life instances in single-track railways within the stringent time limits imposed by the application. The methodology is based on a decomposition of the RTD into two sub-problems: the *Line Dispatching Problem* (LD) and the *Station Dispatching Problem* (SD). The 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 SD problem is the problem of routing and scheduling trains in a station according to a given timetable. The LD problem and the SD problem give raise to distinct sets of variables and constraints. Our approach resembles the classical Benders decomposition

or, better, its combinatorial variant introduced by Codato and Fischetti in [15]. In our decomposition, the LD problem acts as the master problem, whereas the SD problem is the slave. The LD problem is defined on a simplified network, in which each station is represented by a node, and is solved exactly. The solution of the LD problem 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 compatible with the tentative arrival and departure times and is conflict-free. Similarly to [15], 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 stations in the SD problem. As we will show in Section 4, the (general) SD problem 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.

A solution algorithm based on this decomposition approach has been integrated into a semi-automated traffic control system developed by Bombardier Transportation, one of the largest multinational companies in the sector, and is currently operating in a number of single and double-track lines in Italy. However, the current implementation only makes use of simple heuristics to solve the sub-problems of the decomposition. In addition, the LD and SD sub-problems are solved independently, which may even result in infeasible solutions. Indeed, the final decisions are still in the hands of the dispatchers, which may accept or refuse the solutions proposed by the system. The exact algorithm presented in this paper has already been massively tested on instances from the lines mentioned with significant improvements with respect to performances of the system currently in operation. Bombardier Transportation is planning to release an operative implementation of the new approach by the end of 2013: an extensive on-field test-campaign has been scheduled by the end of April 2013 on some relevant single and double-track lines in Central Italy.

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 are able to solve real life instances of single and double-track lines from various regions of Italy within the stringent time limits imposed by the application.
- Our computational results show significant improvements over the performances of the system currently in operation.

2 Problem description

In this section we introduce the main ingredients of the RTD problem. For sake of brevity, we omit here and in the subsequent modeling sections a number of details which are necessary in a practical implementation but irrelevant to describe the crucial modeling and algorithmic issues.

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 a 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 node 0 and 1 are the incoming and exit nodes.

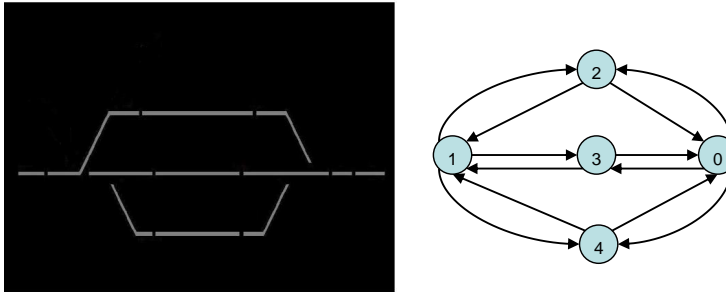


Figure 1: From the station scheme to the oriented graph. Nodes 2, 3, and 4 correspond to platforms

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 time². 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 $V^i = \{v_1^i, v_2^i, \dots\}$. 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.

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_1^i of each route

²In this work we use a fixed speed profile model.

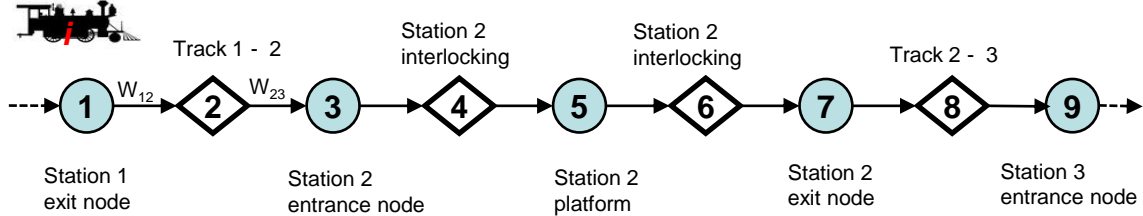


Figure 2: A train route. Circle nodes correspond to tracks preceded by signals, whereas diamond nodes are interlocking routes or tracks between stations.

$R(i)$ with $i \in T$. Each of these arcs has a weight W_{Ov_i} 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 $v = v_k^i$, 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 $v_k^i, v_l^j \in V$ and assume that the rail resource corresponding to v_k^i and the rail resource corresponding to v_l^j cannot be occupied simultaneously. In other words, either train i enters next rail resource v_{k+1}^i on its route before j enters v_l^j , **or** train j enters next rail resource v_{l+1}^j before i enters v_k^i . This can be expressed by the following disjunctive constraint:

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

where we let $t_{x,y} = t_{v_y^x}$ to simplify the notation, and where ϵ 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.

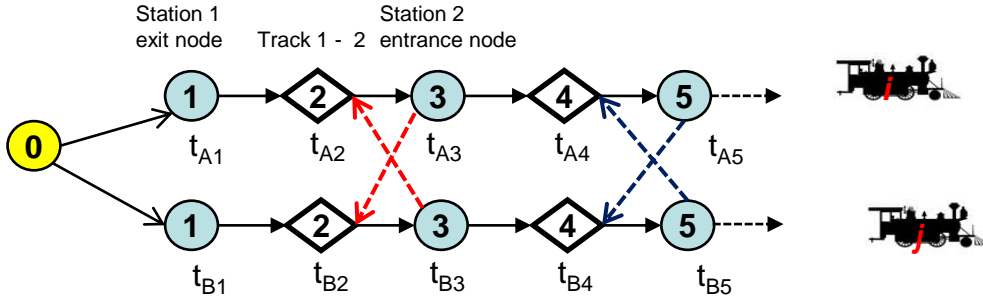


Figure 3: Graph R with disjunctive constraints. Train A cannot enter the track between station 1 and 2 before the train B has entered Station 2, or viceversa

Objective Function The quality of the real-time schedule t depends on its conformity to the official timetable. With engineers from Bombardier Transportation and from the Italian railway network operator we have adopted a convex, piece-wise linear function. Let a_s^i be the arrival time of train $i \in T$ in station $s \in S$. We associate with a_s^i the delay cost function c_s^i 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.

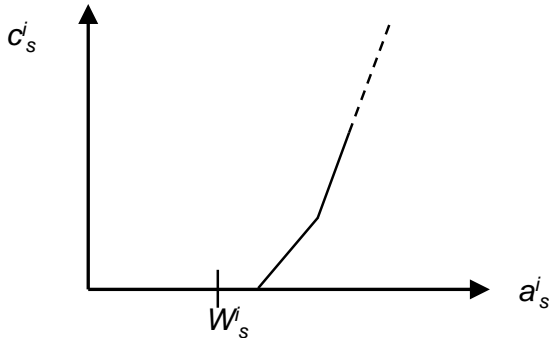


Figure 4: The cost function agreed with practitioners. For train i and station s , W_s^i is the official arrival time, whereas a_s^i is the actual arrival time.

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 problem, we need to solve both a routing and a scheduling problem. The RTD problem can be easily modeled by Mixed Integer Linear Programming (MILP) formulations (as in [28]) or some other techniques to tackle disjunctive programs. However, the RTD instances of practical interest are typically so large that the 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 problem 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.

Single-Track lines We first discuss the case of single-line, single-track railways. "Single-Track" means that there is only one track between any two stations; "Single-Line" means that the infrastructure graph, with vertices corresponding to stations and edges to tracks between stations, is a simple undirected path. The generalization to double-track lines is immediate and will be discussed later. Also the generalization to multi-line railways is straightforward but is out of the scope of the current work and of the real-life instances we are solving in practice. In this case the computational effort increases (depending on the number of lines). Still, exact or approximated decomposition techniques may be applied (see, e.g., [14]). Also, the case of more complex line layouts (i.e. lines with cross-overs) is not addressed here.

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 however 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 [37]. 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 Russia, 47748 km out of 85167 (56%) are single-track [37], while these numbers are even more impressive in China (40257 km out of 66050, or 61%) [37] and India (45237 km out of 64460, or 70%) [20]. 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 [37]. Overall, according to [37], in 2011 single-track lines represented ca. 80% of worldwide railway system. 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 $S = \{1, \dots, q\}$ 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 mentioned in the introduction, we decompose the RTD problem into the real-time *Line Dispatching* (LD) problem, 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 the real-time *Station Dispatching* (SD) problem, a feasibility problem which amounts to finding suitable routes in the station according to 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 problems in the decomposition are not independent of each other. In fact, a solution to the LD problem may result in an inadmissible configuration for the SD problem, as we may not be able to assign station routes to trains as scheduled by the LD problem (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 The real-time 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 $R(i) = \{v_1^i, (v_1^i, v_2^i) \dots, (v_{l(i)-1}^i, v_{l(i)}^i), v_{l(i)}^i\}$ 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 station³. 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 the 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_k^i and v_m^j be the vertices 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:

$$t_{i,k+1} - t_{j,m} \geq \epsilon \tag{3}$$

$$t_{j,m+1} - t_{i,k} \geq \epsilon \tag{4}$$

where ϵ is a positive constant which depends on the infrastructure. Observe that the above precedence constraints correspond to adding to graph R the arcs (v_m^j, v_{k+1}^i) and (v_k^i, v_{m+1}^j) , with weight ϵ . 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.

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 catching up train 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

³Indeed, 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.

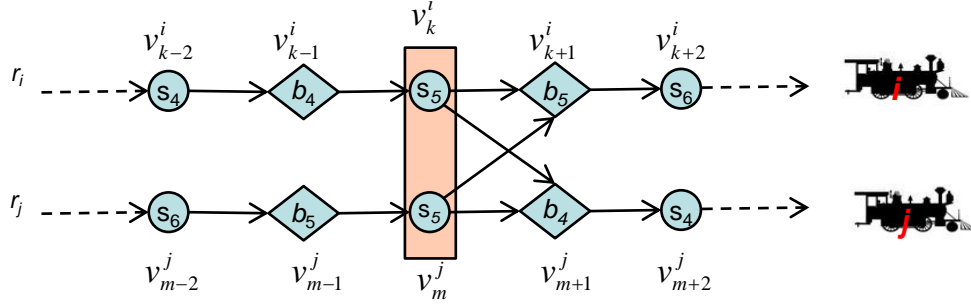


Figure 5: Two trains running in opposite direction and meeting in station s_5 . The movements satisfy the two precedence constraints represented on the graph by two directed arcs.

$s \in S$ and let v_k^i and v_m^j be vertices 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 $t_{j,m-1} - t_{i,k} \geq \epsilon$, $t_{j,m-3} - t_{i,k-2} \geq \epsilon$, ... (i leading before station s), and $t_{i,k+1} - t_{j,m+2} \geq \epsilon$, $t_{i,k+3} - t_{j,m+4} \geq \epsilon$, ... (j leading after station s). Correspondingly, we may represent these constraints on the graph of routes by the set of arcs $A_s^{ij} = \{(v_k^i, v_{m-1}^j), (v_{k-2}^i, v_{m-3}^j), \dots, (v_{m+2}^j, v_{k+1}^i), (v_{m+4}^j, v_{k+3}^i), \dots\}$, as shown in Figure 6.

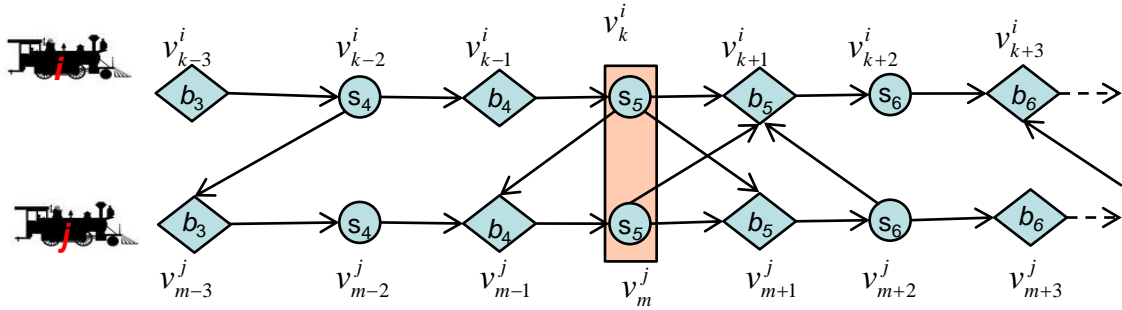


Figure 6: Precedence constraints for two followers meeting in s_5 , represented by arcs on the graph of the routes.

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_s^{ij} of arcs in the graph of the routes R .

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

Then the LD problem can be immediately formulated as follows:

$$\begin{aligned}
\min \quad & c(t) \\
\text{s.t.} \quad & \\
(i) \quad & t_v - t_u \geq W_{uv}, \quad (u, v) \in A \\
(ii) \quad & t_v - t_u \geq M(y_s^{ij} - 1) + \epsilon, \quad (u, v) \in A_s^{ij}, s \in S, \{i, j\} \subseteq T \\
(iii) \quad & \sum_{s \in S} y_s^{ij} = 1, \quad \{i, j\} \subseteq T \\
& t \in \mathbb{R}_+^V, \quad y \text{ binary}
\end{aligned} \tag{5}$$

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.

Lemma 3.1 *Let \bar{y} be a meeting and let $\bar{y}_s \in \{0, 1\}^{\binom{T}{2}}$ be the subvector of y associated with station s . Then \bar{y}_s is the incidence vector of the edges of an interval graph.*

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_s^i 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 $\{Q_s^1, \dots, Q_s^{|T|}\}$. Now, $\bar{y}_s^{ij} = 1$ if and only if $Q_s^i \cap Q_s^j \neq \emptyset$, that is if and only if $\{i, j\} \in E_s$. \square

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 problem 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:

Remark 3.2 *Let (t, y) be a solution to the LD problem. 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 problem (5) cannot in general be extended to a solution of the RTD problem. Indeed, it may be impossible to accommodate trains in stations according to schedule t (which establishes when trains enter and leave the station). The corresponding feasibility problem is the SD problem 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 problem. Any feasible solution (t, y) to the latter will then be feasible also for all the SD problems associated with the 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 tracks⁴. Let $B^D \subseteq B$ be the subset of double-tracks. We introduce a binary variable z_b^{ij} 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

$$\sum_{s \in S} y_s^{ij} + \sum_{b \in B^D} z_b^{ij} = 1 \quad (6)$$

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_b^{ij} on the graph of the routes.

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

$$t_v - t_u \geq M(z_b^{ij} - 1) + \epsilon, \quad (u, v) \in A_b^{ij}, b \in B^D \quad (7)$$

4 Modeling the real-time Station Dispatching problem

We focus now our attention on a station s . A solution to the LD problem provides a timetable for s , that is the time in which each train enters and leaves s . In its more general version, the real-time Station Dispatching Problem (SD) asks for 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 problem closely resembles its off-line version (the *Train Platforming Problem*, see [10]). However, in most practical contexts and in particular in our specific setting, we can make reasonable assumptions that make the problem simpler.

⁴In principle, a follower could catch up and pass the leading train on a double-track section (*parallel run*). However, this manoeuvre is very ticklish and can only be engaged by human operators.

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 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 .⁵ 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 the above assumptions, the SD problem 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 problem.

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 problem 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 $y_s^{ij} = 1$.*

Given a undirected graph $G = (V, E)$, a coloring is a function $c : V \rightarrow N$ 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 $L(i) \subseteq \{1, \dots, k\}$ for $i \in V$, a *list coloring* of G is a coloring c with $c(i) \in L(i)$. Consider a function $\mu : V \rightarrow N$. A μ -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 μ -coloring problem amount to establishing if a graph G admits a coloring, a k -coloring, a list coloring and a μ -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 μ -coloring problems are NP-complete.

It is not difficult to see that the SD problem 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 problem 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 $1, \dots, k$ of the station. We call this case the *all-good* SD problem.

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

⁵Clearly, this does not hold for larger stations, where several routes go through the same platform.

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

The platforms of a station may be partitioned according to the incoming direction of the 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, \dots, T_k be a partition of the trains T and let P_1, \dots, P_k be a partition of the platforms P . Assume that a train in T_q can access all platforms in P_q , $q = 1, \dots, k$, and no other platforms. Then the corresponding SD problem 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 problem, 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 problem. We have that:

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

Proof. Reduction from μ -coloring on unit interval-graphs. A unit interval graph is the intersection graph of unit length intervals. Observe that every μ -coloring uses at most $k_\mu = \max_{i \in V} \mu(i)$ colors. So, given the function μ , and a unit interval graph $H = (V, E)$ we construct an instance of the hierarchical SD problem in the following way. We consider a single station line. We let the set of trains $T = V$, the platforms $P = \{1, \dots, k_\mu\}$ 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 problem is feasible, and let $c : T \rightarrow P$ be an assignment of platforms to trains. Then c is also a μ -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 μ -coloring of H . \square

An alternative way to derive the above complexity results is to exploit the relation between the (no routing) SD problem and the *Interval Scheduling* problem (see [21]).

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 problem is equivalent to this basic version of Interval Scheduling. [21] also introduces the *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 problem.

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 [39]. Nevertheless, the decomposition principle here introduced along with the master-slave solution approach are still exploitable.

4.1 MILP models for the SD problem

Basically all the instances in our (real life) test-bed belong to the all-good SD problem, with exceptions which can be handled easily. In what follows we discuss two different approaches to the solution of the all-good SD problem. 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 *station capacity*.

A compact, flow based representation of the all-good SD problem. Our purpose is to "embed" the all-good SD problem in (5) in order to derive a MILP for the RTD problem. To this end, we need to express the all-good SD problem 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 problem, 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 $s \in \{1, \dots, |S|\}$, the quantity x_s^{ij} 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 $1 \leq k \leq |S|$ ($y_k^{ij} = 1$), 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:

$$x_s^{ji} = \sum_{q < s} y_q^{ij}, \quad s \in S$$

and

$$x_s^{ij} = 1 - \sum_{q <= s} y_q^{ij}, \quad s \in S$$

Similar transformations may be derived for a pair of followers. In general, there exists an affine transformation from y to x , i.e.

$$x = Qy + q \tag{8}$$

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 problem 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 \bar{y} , along with the corresponding successors vector \bar{x} . For sake of simplicity, we assume that every train in T goes through s . We introduce the support graph $N(s, \bar{x}) = (\{r, p\} \cup U \cup W, E)$, where $U = \{u_1, \dots, u_{|T|}\}$, $W = \{w_1, \dots, w_{|T|}\}$. 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 $e \in E_r \cup E_U \cup E_p$, $f_{(w_i, u_j)} = \bar{x}_s^{ij}$ 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.

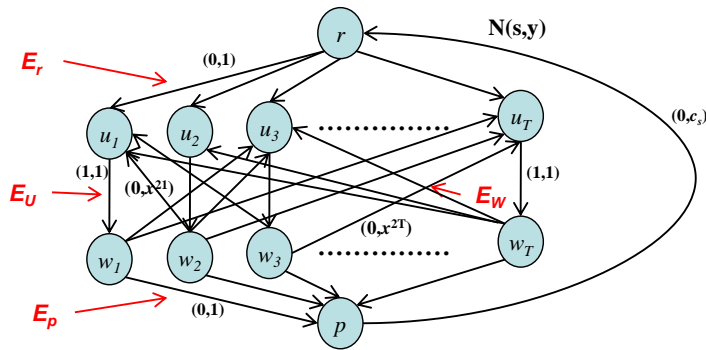


Figure 7: The support graph. Lower and upper bounds are shown between brackets for some representative arcs

We have the following

Theorem 4.7 *The all-good SD problem has a solution if and only if the graph $N(s, \bar{x})$ 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 problem. 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 for station capacity. Consider a station $s \in S$ with c_s platforms and let (t, y) be a solution to the LD problem. 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, $\sum_{\{i,j\} \subseteq K} y_s^{ij} = \frac{1}{2}(c_s + 1)c_s$. In other words, the meeting y does not violate station capacity if and only if, for all $s \in S$, we have:

$$\sum_{\{i,j\} \subseteq K} y_s^{ij} \leq \frac{1}{2}(c_s + 1)c_s - 1 \quad (9)$$

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

5 Solution Algorithm

We are finally able to formulate the RTD problem as a Mixed Integer Linear Program by coupling constraints (9) and program (5) and linearizing the objective function:

$$\begin{aligned}
\min \quad & c(t) \\
s.t. \quad & \\
(i) \quad & t_v - t_u \geq W_{uv}, & (u, v) \in A \\
(ii) \quad & t_v - t_u \geq M(y_s^{ij} - 1), & (u, v) \in A_k^{ij}, k \in S, \{i, j\} \subseteq T \\
(iii) \quad & t_v - t_u \geq M(z_b^{ij} - 1), & (u, v) \in A_b^{ij}, b \in B^D, \{i, j\} \in T, i, j \text{ crossing} \\
(iv) \quad & \sum_{s \in S} y_s^{ij} + \sum_{b \in B^D} z_b^{ij} = 1, & \{i, j\} \subseteq T \\
(v) \quad & \sum_{\{i, j\} \subseteq K} y_s^{ij} \leq \frac{1}{2}(c_s + 1)c_s - 1, & s \in S, K \subseteq T, |K| = c_s + 1 \\
& t \in \mathbb{R}_+^V, \quad y, z \text{ binary}
\end{aligned} \tag{10}$$

To simplify the notation, we write constraint (10) in the same form every pair of trains, by assuming $z_b^{ij} = 0$ for all pair 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 grow exponentially with the number of trains and the capacity of the stations. Also, the number of constraints (10.ii) can grow 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's decomposition algorithm (see, e.g., [31]), a relaxed problem is solved in every node of the branching tree and Bender's cuts violated by the current fractional solution are generated. In contrast, in their combinatorial variant, Codato and Fischetti prefer to solve to (integral) optimality the original master problem; then, violated combinatorial Bender's cuts are generated and added, and the revised master problem is again solved to integral optimality. We follow a somehow intermediate path. Rather than generating violated constraints of type (10.v) in every node of the branching tree, we limit to the nodes corresponding to integer solutions. In this way, the slave problem is precisely the SD problem 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 [18]). On the other hand, we do not need to solve several integer problems to optimality as in [15].

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 [26]), leaving very little hope to solve the separation efficiently.

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

All our real life cases satisfy the assumptions of the all-good SD problem. Once a feasible meeting y is found, it is immediate to obtain a platform assignment by coloring the interval graphs $G(y_s)$ for all $s \in S$. Incidentally, 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 problem introduced in Section 4 suitable cuts in the y variables still suffice to represent infeasibility.

In order to provide an initial upper bound to the solution process we developed a simple, heuristic algorithm, sketched next. Again, we decompose the problem into a LD problem and a SD problem: however the optimization sub-problems are solved heuristically according to the prioritization rules of the Italian Railway Operator. In particular, at each iteration, potentially conflicting pairs of trains are identified and ordered chronologically. The first conflict in this ordering is then *solved* by establishing, for the corresponding pair of trains, a meeting point, either a station or a double-track (if a given choice would result in capacity violation, it is disregarded). For any choice, one of the two trains must wait some time for the other. The final chosen point is the one which minimizes such waiting time. Once the conflict is solved, the 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 never happened in our experiments on real-life instances.

A practical implementation of the above heuristic algorithm is currently operating on several railway lines in Italy (see Section 6)⁶.

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 (with no intervention) 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. The dispatchers then force drivers to follow their decisions by switching suitable traffic signals to red light. Adding a constraint of type (10.ii) is the mathematical equivalent to activating a red signal.

⁶In the current version the solutions generated are displayed to the dispatchers, which can accept them or refuse them.

6 Computational Results

The objective of our computational tests is twofold. Firstly, we want to identify the most effective approach between the decomposition and the compact formulation to solve the RTD problem. Secondly, we confront the best approach, namely the decomposition, with current practice. We 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. Details about these lines are given in Table 1.

Line	Abbr.	Stops	Stations	Length (m)	S.T.	D.T.
Trento - Bassano	T-BG	22	14	95711	yes	no
Piraineto - Trapani	P-T	12	12	93532	yes	no
Alcamo - Trapani	A-T	14	13	116119	yes	no
Terontola - Foligno	T-F	18	11	82200	yes	no
Foligno - Orte	F-O	13	10	82018	yes	yes
Falconara - Foligno	F-F	24	17	119612	yes	yes

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

All instances were provided by Bombardier Transportation, one of the key global actors of the railway sector. As mentioned, a simplified version of the approach has been already developed in cooperation with Bombardier Transportation, and is currently operating on these lines.

The instances in our test-bed were extracted at peak hours and refer to existing trains actually running on the above lines. We fixed a 60 second time-limit for our tests, which is regarded as an acceptable time span for a dispatcher’s decision making. In practice, real life requirements are often less stringent⁷.

Implementation details. Tests on instances in Table 2, 3, 4 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(R) Core(tm) i7-2640M CPU 870 2.80GHz machine using CPLEX 12.2. To implement Row Generation within branching nodes, CPLEX offers different strategies. An extension of the LazyConstraintCallback class was used: the cut separator is called only if the current solution is integer. The recommended settings for CPLEX were: (i) Turning off presolve, (ii) Avoiding multiple threads (iii) Setting the MIP search to Primal (iv) Turning off dual reduction. In addition we also disabled the heuristic search ($HeurFreq = -1$) and increased integer tolerance.

⁷This much depends on distances between stations. Norwegian dispatchers allow, for instance, up to 10 minutes reaction time for some single-track lines [33]

6.1 Confronting compact formulation against 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, as we will see, the compact formulation proved to be less effective than the non compact one. A summary of computed results is shown in Table 2 and Table 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.

ID	Line	Trains	Initial Rows		Generated		Computation Time(s)	
			C	NC	C	NC	C	NC
1	T-BG	29	9459	1333	212	177	3,99	0,58
2	T-BG	28	8980	1302	246	130	8,25	0,78
3	T-BG	29	9691	1351	124	93	1,98	0,54
4	T-BG	29	4794	912	762	299	10,16	0,83
5	T-BG	22	4858	918	688	453	9,09	2,69
6	P-T	24	3781	898	741	259	2,98	0,98
7	P-T	24	3530	873	790	323	8,10	0,84
8	P-T	24	3553	882	311	516	1,38	1,26
9	P-T	23	3417	862	313	412	1,31	1,09
10	P-T	26	3669	904	399	189	6,81	1,09
11	A-T	20	4031	832	975	435	32,57	0,97
12	A-T	20	4250	851	661	386	5,74	11,04
13	A-T	19	3764	832	707	743	10,14	5,81
14	A-T	19	3729	796	590	763	22,25	6,13
15	A-T	18	3057	755	830	786	28,08	6,41
Average							5,53	1,34

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.

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. Figure 8 shows the evolution of the integral solution values and of the gap (between the incumbent solution and the current LB) for one instance of the

Alcamo - Trapani line.

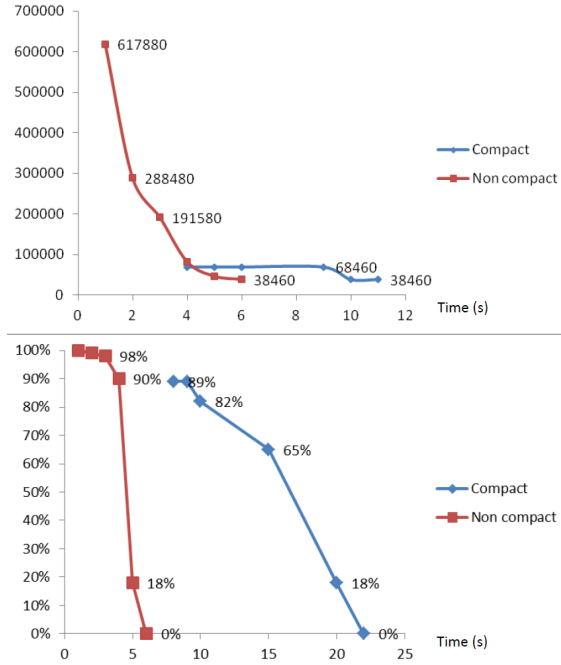


Figure 8: Above, evolution of feasible solutions value; Below, evolution of the gap

In other cases, namely for some instances of the Trento - Bassano line, 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. We report the number of controlled trains (column "Trains"), the best solution found and gap values for increasing time. In most cases, the non-compact formulation produced better solutions and terminated with a gap which was at most 52%. Only for one instance the compact formulation terminated with a better solution. In another case, however, the compact formulation was not able to produce a feasible solution at all.

In these test instances, the algorithm covers an average 9 to 13 hour time horizon. Consequently, re-routing and re-scheduling decision making may often take in account, unlike dispatchers, trains which are not due on the line for many hours to come. In this sense, a reasonable approach could be narrowing the time window and, consequently, dropping a few of the last trains (according to their entry times on the line), precisely the number that allows to find the optimal solution within the time limit. In table 4, we report, for each instance of Table 3, the number of trains remaining and, between brackets, the number of trains removed.

ID	F	Trains	10 s		30 s		60 s		180 s		300 s	
			gap	sol	gap	sol	gap	sol	gap	sol	gap	sol
i1	C	19	93%	1575	93%	1575	80%	1575	70%	1575	69%	1575
i1	NC	19	0%	1219	0%	1219	0%	1219	49%	1219	0%	1219
i2	C	28	-	-	48%	1799	48%	1799	48%	1799	48%	1799
i2	NC	28	57%	2200	57%	2200	57%	2200	22%	1262	18%	1259
i3	C	28	-	-	24%	1266	24%	1266	20%	1266	15%	1266
i3	NC	28	53%	2153	43%	1766	35%	1546	33%	1546	33%	1546
i4	C	28	-	-	-	-	56%	2765	56%	2765	51%	2765
i4	NC	28	36%	1922	34%	1922	33%	1922	33%	1922	32%	1922
i5	C	28	-	-	-	-	-	-	-	-	-	-
i5	NC	28	54%	3307	53%	3307	53%	3307	52%	3307	52%	3307

Table 3: Instances where the algorithm could not prove optimality within time limits (60 seconds) for both formulations.

ID	F	Trains	Time
i1	C	12 (7)	0,80
i2	C	9 (19)	13,85
i2	NC	10 (18)	8,22
i3	C	9 (19)	14,63
i3	NC	9 (19)	5,69
i4	C	7 (21)	0,94
i4	NC	8 (20)	11,67
i5	C	7 (21)	1,09
i5	NC	8 (20)	7,30

Table 4: Reduced instances: the solution is found within 15 seconds

6.2 Evaluating the decomposition approach

The former tests show that 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. We verified that

dispatchers follow the decisions taken by the heuristic algorithm more than 90% of times. Also, 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 each Table 5, 6 and 7, we present a summary of computed results for 10 representative instances from single and double-track lines in Italy. Note that E stands for *Exact* and H stands for *Heuristic* and that, again, the time limit was fixed to 60 seconds.

# Trains	Solution		Optimal		Gap		# Conflicts		Time(s)
	E	H	E	H	E	H	E	H	E
45	696	1010	yes	no	0%	31%	13	13	0,55
46	494	1397	yes	no	0%	65%	16	19	1,38
48	203	293	yes	no	0%	31%	20	38	7,53
50	277	314	yes	no	0%	12%	15	14	3,26
49	360	422	yes	no	0%	15%	17	14	26,89
50	263	346	yes	no	0%	24%	22	25	20,51
50	593	653	yes	no	0%	9%	24	27	10,79
50	691	759	yes	no	0%	9%	30	31	33,61
50	180	210	no	no	13%	14%	15	17	> 60
50	690	690	no	no	15%	15%	30	38	> 60

Table 5: 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 (with consequent calls to the Separator), for the heuristic and exact algorithm, respectively.

For the instances in Tables 5, 6, 7 we computed the highest number of trains on-line simultaneously, which was 8, 11 and 7, respectively. However, although dispatching on-line trains has an immediate impact on network congestion, handling only such trains is not sufficient to effectively tackle the RTD problem. In the general case, each on-line train also interacts with a number of trains which will enter the line before the former leaves it. For instance, the on-line trains mentioned above could have suffered conflicts with other 6, 12 and 6 distinct trains, respectively, which were off-line at the time. Indeed, in any case of practical interest, the interval graph associated with train arrivals and departures from a railway line is a connected graph. In other words, each dispatching decision has an effect not only locally, but propagates on the network and may cause severe alterations over time.

In most cases, the exact algorithm finds optimal solutions within the time limit. In other cases, good feasible solutions were found. It is interesting to notice that, even

# Trains	Solution		Optimal		Gap		# Conflicts		Time(s)
	E	H	E	H	E	H	E	H	E
33	494	576	yes	no	0%	14%	34	21	11,22
34	958	1678	yes	no	0%	43%	16	15	30,62
34	261	972	yes	no	0%	73%	15	16	35,89
34	625	1345	yes	no	0%	43%	15	16	7,36
51	636	878	no	no	73%	80%	59	65	> 60
51	497	523	no	no	71%	74%	58	44	> 60
51	585	632	no	no	64%	67%	50	44	> 60
60	985	985	no	no	31%	31%	101	56	> 60
56	924	1139	no	no	18%	33%	74	50	> 60
69	1280	1280	no	no	85%	85%	73	101	> 60

Table 6: Computational results for the Falconara - Foligno 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.

when optimality is not proven, the quality of such solutions is generally higher than the corresponding heuristic ones.

To highlight the impact of dispatching decisions on the real-time timetable, in Table 8 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 by general public. In Table 8, we report the average distribution of delayed trains for the exact and heuristic algorithm, computed by solving 500 instances for each of the relevant single and double track lines. Based on feedback from the railway operators, possible delays were subdivided in three macroscopic ranges: on-time (less than 3 minutes), 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.

As emerges from Table 8, 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 slightly less trafficked lines, with an increase in the number of trains on time of as much as 26% for the Trento-Bassano line and 24% for the Terontola-Foligno line, while still noticeable for the two more complex lines, 4% for the Falconara-Foligno line and 9% for the Foligno-Orte line. Also, Table 8 shows how the average improvement in punctuality is not only due to slightly delayed trains arriving on time, but also to a clear decrease in the number of trains running severely late.

However, the authors point out that although, for the more complex lines, there may seem to be less "distance" between exact and heuristic algorithms (in terms of solution quality), this is actually a bias caused by the non negligible amount of times the exact

# Trains	Solution		Optimal		Gap		# Conflicts		Time(s)
	E	H	E	H	E	H	E	H	E
38	123	366	yes	no	0%	66%	48	41	52,18
41	124	369	yes	no	0%	66%	49	46	41,66
42	130	375	yes	no	0%	65%	51	45	40,12
45	185	372	yes	no	0%	50%	36	36	31,87
45	189	254	yes	no	0%	26%	34	30	18,19
44	135	332	yes	no	0%	59%	33	38	8,16
43	194	348	yes	no	0%	44%	33	38	7,32
43	143	393	no	no	12%	68%	51	48	> 60
44	370	498	no	no	46%	60%	45	54	> 60
44	541	630	no	no	30%	40%	37	52	> 60

Table 7: Computational results for the Terontola - Foligno line (single-track), over an average 13 hour horizon. Note that, as heuristic computation times are always under a second, they have been omitted.

Line	# Trains	Horizon (hrs)	On Time		Late between 3 and 6 mins		Later than 6 mins	
			H	E	H	E	H	E
T-BG	29	9	49%	75%	12%	3%	39%	22%
T-F	36	11	65%	89%	24%	7%	11%	4%
F-O	40	12	83%	92%	9%	3%	8%	5%
F-F	54	12	77%	81%	9%	7%	14%	12%

Table 8: Punctuality distribution for 500 instances. Average figures.

algorithm failed to prove optimality (within the time limit). In Table 9 we limit our statistics to instances for which the final gap is 0 (i.e. the optimal solution is found and proven). The figures show that in this case the impact on punctuality/delay distribution is very similar also for the more complex, denser lines. Indeed, in the general picture, it is perhaps precisely for these lines that using an exact approach would produce the most measurable and beneficial impact. In our opinion, the results in Table 9 confirm, furthermore, the relevance of developing effective exact methods for the RTD problem.

Over all, our results show how the implementation of the new approach will significantly increase the quality of the real-time plan with respect to the current practice. As mentioned, the new approach is scheduled to be put in operation as of April 2013, for an extensive test-campaign, on the three single-double track lines centered in Foligno.

Line	% Gap 0	On Time		Late between 3 and 6 mins		Later than 6 mins	
		H	E	H	E	H	E
T-BG	47%	58%	90%	12%	3%	31%	7%
T-F	85%	66%	91%	22%	7%	12%	3%
F-O	95%	80%	89%	10%	5%	10%	7%
F-F	35%	75%	87%	13%	5%	13%	7%

Table 9: Punctuality distribution for instances with gap 0%. Column "% Gap 0" indicates the percentage of instances for which the final gap is 0. Average figures.

7 Current developments and acknowledgments

A number of research directions are currently being explored. We are investigating the possibility of strengthening our relaxation by identifying stronger valid inequalities. We are looking for new branching strategies. We are also studying alternative ways to represent conflicts. Finally, we are extending the approach to deal with multiple routes and more complex station layouts.

We wish to thank for their precious contributions and assistance: Salvatore Caranante, Ferruccio Fiorucci, Alessandro Mascis and Franco Pietrini from Bombardier Transportation, Paolo Perticaroli and Mauro Piacentini from the railway division of STAER Sistemi (Italy), Thomas Nygreen from the Capacity Department of the Norwegian network operator (Jernbaneverket), Arnt Rogstad from Trondheim Dispatching Central.

References

- [1] Ahuja, R.K., T.L. Magnanti, J.B. Orlin, *Network Flows*, Prentice-Hall, 1993.
- [2] Alvrás D. and Padberg M.W., *Linear Optimization and Extensions: Problems and Soluzioni*. Springer-Verlag, Berlin, Germany, 2001.
- [3] Balas, E., Machine sequencing via disjunctive graphs, *Operations Research* 17 (1969) pp. 941–957.
- [4] Balas, E., *Disjunctive programming*, *Annals of Discrete Mathematics*, **5**, pp. 3–51, 1979.
- [5] F. Bonomo, G. Duran, J. Marenco, *Exploring the complexity boundary between coloring and list-coloring*, *Annals of Operations Research* 169(1), pp. 3–16, 2009.

- [6] F. Bonomo, Y. Faenza, G. Oriolo *On coloring problems with local constraints*, Discrete Mathematics, 312(12-13) pp. 2027-2039, 2012.
- [7] Borndörfer R., T. Schlechte, *Models for Railway Track Allocation*, ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Eds. Christian Liebchen and Ravindra K. Ahuja and Juan A. Mesa.
- [8] Brännlund, U., P.O. Lindberg, A. Nou, J.-E Nilsson, *Railway Timetabling using Lagrangian Relaxation*, Transportation Science, Vol 32 (4), pp. 358-369, 1998.
- [9] Caprara, A., M. Fischetti, P. Toth, *Modeling and solving the train timetabling problem*, Operations Research, 50 (5) 292, pp. 851-861, 2002.
- [10] A. Caprara, L. Galli, P. Toth. *Solution of the Train Platforming Problem*, Transportation Science, 45 (2), pp 246-257, 2011.
- [11] A. Caprara, L. Kroon, M. Monaci, M. Peeters, P. Toth, “Passenger Railway Optimization”, in C. Barnhart, G. Laporte (eds.), *Transportation*, Handbooks in Operations Research and Management Science 14, Elsevier (2007) 129–187.
- [12] M.C. Carlisle, E.L. Lloyd, *On the k -coloring of intervals*, Discrete Applied Mathematics, 59, pp 225–235, 1995.
- [13] F. Corman, A. D’Ariano, D. Pacciarelli, M. Pranzo *A tabu search algorithm for rerouting trains during rail operations*, Transportation Research Part B 44 (2010) 175–192
- [14] Corman, F., D’Ariano, A., Pacciarelli, D., Pranzo, M. *Optimal inter-area coordination of train rescheduling decisions*. Transportation Research E, Logistics and Transportation Review, 48 (1), 71-88.
- [15] Codato G., Fischetti M., *Combinatorial Benders’ Cuts for Mixed-Integer Linear Programming*, Operations Research, 54 (4), pp. 756-766, 2006.
- [16] Dyer., M., L. Wolsey, *Formulating the single machine sequencing problem with release dates as a mixed integer program*, Discrete Applied Mathematics, no. 26 (2-3), pp. 255-270, 1990.
- [17] Goldberg, A.V., R. Tarjan, *Finding minimum cost circulation by successive approximation*, Math. of Op. Res., 15, pp. 430-466, 1990.
- [18] U.I. Gupta, D.T. Lee, J.Y.T. Leung, *Efficient algorithms for interval graphs and circular-arc graphs*, Networks 12, pp.459–467, 1982.
- [19] S. Harrod, *Modeling Network Transition Constraints with Hypergraphs*, Transportation Science 45 (1), pp. 81-97, 2011

- [20] Indian Railway *Year Book 2010-11*, pp.22
- [21] A.W.J. Kole, J.K. Lenstra, C.H. Papadimitriou, F.C.R. Spijksma, *Interval Scheduling: A Survey*, Naval Research Logistics, 54, pp. 530–543, 2007.
- [22] L. Kroon, D. Huisman, E. Abbink, P.-J. Fioole, M. Fischetti, G. Maroti, A. Schrijver, A. Steenbeek, R. Ybema, *The New Dutch Timetable: The O.R. Revolution*, Interfaces 39 (1), pp. 6-17, 2009
- [23] Jernbaneverket Presentation 2009, http://www.jernbaneverket.no/PageFiles/7535/JBV-presentasjon_web_2009_05_07.pdf
- [24] L. Lamorgese, C. Mannino, An exact decomposition approach for the real-time train dispatching problem, Technical Report N. A23274, SINTEF ICT, Norway, 2012, submitted.
- [25] Luethi M., *Improving the Efficiency of Heavily Used Railway Networks through Integrated Real-Time Rescheduling*, Ph. D. Thesis, ETH Zurich, 2009.
- [26] E.M. Macambira, C.C. de Souza, *The edge-weighted clique problem: Valid inequalities, facets and polyhedral computations*, European Journal of Operational Research, 123 (2), pp. 346–371, 2000.
- [27] Mascis A., *Optimization and simulation models applied to railway traffic*. Ph.D. thesis, University of Rome “La Sapienza”, Italy, 1997. (In Italian).
- [28] C. Mannino, A. Mascis, *Real-time Traffic Control in Metro Stations*, *Operations Research*, 57 (4), pp 1026-1039, 2009
- [29] Mascis A., D. Pacciarelli, *Job shop scheduling with blocking and no-wait constraints*, European Journal of Operational Research, 143 (3), pp. 498–517, 2002.
- [30] M. Montigel, *Semi-Automatic Train Traffic Control in the New Swiss Lötschberg Base Tunnel*, IRSA-Apect 2006, www.systransis.ch/fileadmin/2006_Paper_MM.pdf
- [31] Nemhauser G.L. and Wolsey L.A., *Integer and Combinatorial Optimization*, Wiley-Interscience, 1999.
- [32] Rete Ferroviaria Italiana, <http://www.rfi.it/>.
- [33] Rogstad A.E., Chief Dispatcher Trondheim station, Norway *Personal Communication*, 2013.
- [34] G. Sahin, R.K. Ahuja and C.B. Cunha, *Integer Programming Based Approached for the Train Dispatching Problem*, Tech. Rep. Sabanci University, 2010.

- [35] A. Schrijver, *Combinatorial Optimization*, Springer, 2003.
- [36] J. Törnquist, J. A. Persson, *N-tracked railway traffic re-scheduling during disturbances*, Transportation Research Part B 41, 342–362, 2007.
- [37] International Union of Railways, *Synopsis 2011*
- [38] Zwaneveld, P.J., Railway Planning. Ph.D. Thesis, Rotterdam School of Management, TRAIL, Rotterdam, 1997.
- [39] Zwaneveld, P.J., L.G.S. Kroon, H.E. Romeijn, M. Salomon, S. Dauzere-Peres, S.P.M. Van Hoesel, H.W. Ambergen, *Routing trains through railway stations: model formulation and algorithms*, Transportation Science, 30 (3), pp. 181-194, 1996.

8 Appendix

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

Proof.

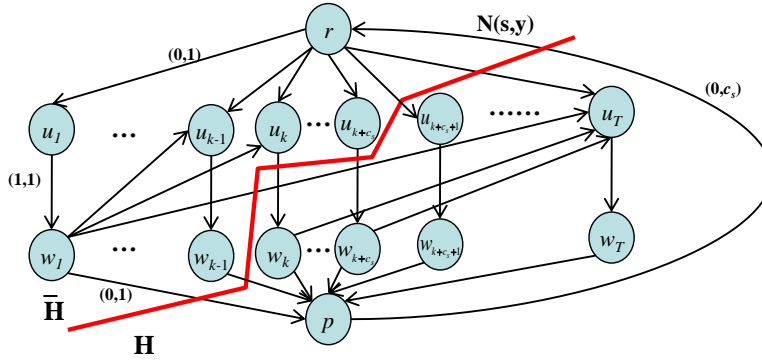


Figure 9: The cut in the necessity proof of Theorem 4.7. We dropped from the figure all arcs of type (w_j, u_i) with $i < j$ (i not successor of j)

By contradiction, we assume that $N(s, \bar{x})$ has a circulation but the all-good SD problem 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 $Q = \{k, \dots, k + c_s\} \subseteq T$, which are simultaneously in station s . We construct a cut by letting $H = \{p\} \cup \{w_j : j = \{k, \dots, |T|\} \cup \{u_j : j = \{k + c_s, \dots, |T|\}\}$.

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 $(u_k, w_k), \dots, (u_{k+c_s}, w_{k+c_s})$ (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 $j = k + c_s, \dots, |T|$. But then also $w_j \in H$ for $j = k + c_s, \dots, |T|$.
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 $j \in \{1, \dots, k, \dots, k + c_s\}$ 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 $Q = \{k, \dots, k + c_s\}$ 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 .

□



Technology for a better society

www.sintef.no