A real-time conflict solution algorithm for the train rescheduling problem

Bettinelli, Andrea; Santini, Alberto; Vigo, Daniele

published in
Transportation Research. Part B, Methodological
2017

DOI (link to publisher)
10.1016/j.trb.2017.10.005

document version
Publisher's PDF, also known as Version of record

Transportation Research Part B 106 (2017) 237–265

Contents lists available at ScienceDirect

Transportation Research Part B

journal homepage: www.elsevier.com/locate/trb

A real-time conflict solution algorithm for the train rescheduling problem

Andrea Bettinelli[a], Alberto Santini[b], Daniele Vigo[c,a,*]

[a] Optit s.r.l., Viale Amendola 56/D, 40026 Imola (BO), Italy
[b] Departament d'Economia i Empresa, Universitat Pompeu Fabra and Barcelona GSE, Trias-Fargas 25, 08005 Barcelona, Spain
[c] DEI, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy

ARTICLE INFO

Article history:
Received 4 March 2016
Revised 7 October 2017
Accepted 9 October 2017
Available online 28 October 2017

Keywords:
Railway optimization
Train rescheduling
Real-time algorithm
Heuristic

ABSTRACT

We consider the real-time resolution of conflicts arising in real-world train management applications. In particular, given a nominal timetable for a set of trains and a set of modifications due to delays or other resources unavailability, we are aiming at defining a set of actions which must be implemented to grant safety, e.g., to avoid potential conflicts such as train collisions or headway violations, and restore quality by reducing the delays. To be compatible with real-time management, the required actions must be determined in a few seconds, hence specialized fast heuristics must be used.

We propose a fast and effective parallel algorithm that is based on an iterated greedy scheduling of trains on a time-space network. The algorithm uses several sortings to define the initial train dispatching rule and different shaking methods between iterations. The performance is further enhanced by using various sparsification methods for the time-space network. The best algorithm configuration is determined through extensive experiments, conducted on a set of instances derived from real-world networks and instances from the literature. The resulting heuristic proved able to consistently resolve the existing conflicts and obtaining excellent solution quality within just two seconds of computing time on a standard personal computer, for instances involving up to 151 trains and two hours of planning time horizon.

1. Introduction

Modern railways represent a major form of transport with an ever-growing user base, as trains are flexible in terms of travelling distance (they can be used for local, regional and long-distance services) and capacity (as they are modular by nature). Furthermore, train transportation is usually the greenest transportation options for both goods and people.

Despite this, railways are confronted with the increase of operational costs and a fierce competition from other modes of transport. Many users demand more reliability in train operations: a long delay, a cancelled train, a missed connection can easily decrease the perceived quality of service and turn away potential customers.

Most of the events that negatively affect train operations (broadly called *conflicts*) happen when, for some reason, there is a difference between the nominal and the actual service. The causes of such events are usually divided into *disturbances* and *disruptions* (Cacchiani et al., 2014). The former are small perturbations of the system that are handled by network operators

* Corresponding author.
E-mail addresses: andrea.bettinelli@optit.it (A. Bettinelli), alberto.santini@upf.edu (A. Santini), daniele.vigo@unibo.it (D. Vigo).

https://doi.org/10.1016/j.trb.2017.10.005
0191-2615/

by momentarily changing the timetable. The results of disturbances are usually minor, such as one or more delayed trains, or a platform change at a station. Disruptions, on the other hand, are major incidents that not only alter the nominal timetable, but also require changes in rolling stock and crews. The outcome of a disruption could include major delays, train cancellations, and long reroutings. Disturbances clearly happen much more often than disruptions and their impact is not to be underestimated: a train that is delayed just a few minutes can make a user miss an important connection and increase their travel time by hours. In this paper we consider both disturbances and disruptions in a unified way, by defining an algorithmics approach to handle the conflicts they cause.

Increasing systemwide reliability is crucial at every phase of the planning process. It starts at the strategic and tactical levels (budget allocation for maintenance, timetable robustness, etc.), but once at the operational level, it is almost impossible to avoid that day-to-day activities be disturbed by many kinds of unforeseen events.

When such an event occurs, it is the job of the *dispatcher* to restore the system in a working state. The job of dispatchers has been traditionally done by hand, based exclusively on their experience and practice. It was not until recent years that computer algorithms were developed with the aim of aiding the dispatchers in making the best decision that resolves the critical situation and minimises deviances from the nominal timetable.

In this paper we present such an algorithm, developed to solve the Train Rescheduling Problem (TRP): given a nominal timetable which has become infeasible because of one or more *conflicts* that have arisen, we are asked to produce a new conflict-free timetable that is as close as possible to the nominal one. Or, in case it is not possible to produce a conflict-free timetable, we need to warn the dispatcher about this and provide a timetable with the least possible number of conflicts.

Conflicts are all those situations that either can't physically happen (e.g., two trains occupying the same segment of track at the same time) or that can potentially compromise the safety of operations in the network (e.g., two trains running too close to each other in the same direction).

The algorithm presented in this paper is the result of a long lasting collaboration with Alstom, initiated by the company in 2012 with the aim of redesigning the optimisation algorithms incorporated in its Train Management System ICONIS. To this end, Alstom involved three important Italian research groups in specific research projects investigating various optimisation problems arising in the real time conflict resolution. As a result of such initial wide research effort, the team formed by Optit, an accredited spinoff of the University of Bologna, and the Department of Electrical, Electronic and Information Engineering of the University of Bologna, was selected to produce an innovative real-time conflict solution algorithm capable of taking into account the characteristics and constraints of practical applications which has been developed and industrialised during 2013, and extensively tested by Alstom in real-world contexts. Recently, the new algorithm has been fully integrated in ICONIS and will be deployed at various international Alstom customers.

The paper is structured as follows. In the next section we give an overview of how a railway system works, how it can be affected by disturbances and what it means to reschedule a train. In Section 3 we review the existing literature on the TRP, based on the classification schema given by Cacchiani et al. (2014). In Section 4 we give a mathematical description of a railway network, of train timetables and of the relationship between them. We present an heuristic algorithm for the solution of the TRP in Section 5. We then describe the instances used and provide computational results in Section 6. Finally, we draw conclusions and propose further research paths in Section 7.

2. Timetables and conflicts

Nominal timetables are the crucial part of any railway systems. They describe in detail the trip of each train, from its departure to its arrival station, including all the intermediate stations where the train stops or passes by. This includes not only those parts of the trip where the train operates passenger service, but also all the movements necessary to perform service and maintenance, e.g., rolling stock relocation, cleaning, technical service.

Every arrival and departure is scheduled at specific time slots, which are calculated in advance by taking into account physical properties (e.g., track curvature and gradient, maximum allowed speed, train length) and interaction among trains. Clearly two trains can't occupy the same portion of tracks at the same time, but other constraints usually have to be respected. For example trains have to respect *headway times*, i.e., a minimum amount of time must be left as a buffer between trains travelling in the same direction. Another example are *dwell times* at platforms, which are needed to board and alight passengers.

Timetables can be *periodic* or *aperiodic*. Periodic timetables repeat themselves at certain time intervals (e.g., every second hour and every hour during peak times). Although such timetables are usually appreciated by customers, as they are easy to memorise and use, they are difficult to implement in a competitive market where many train operators are likely to request access to the same resources at the same time. For this reason, trains are often scheduled in aperiodic timetables. The name *aperiodic* is slightly misleading, since these timetables are repeated day after day so, strictly speaking, they have a period of one day.

Timetables are implemented by assigning *tasks* to *rolling stock* and *crews*. When it comes to passenger transportation, rolling stock are usually composed of one or more locomotives and many passenger cars; or, in case of multiple unit (MU) trains (MU trains are those composed by one or more similar self-propelled train cars), by one or more MUs. A crew includes a train driver and one or more train guards. Finally, a task represents a complete trip of the rolling stock and the crew from the train origin to its destination. The set of tasks carried out by rolling stock and crews in a day is called a *shift*, or duty.

Since in most countries the railway infrastructure is operated by a different actor than the trains, the timetables are usually created and managed by an *infrastructure manager*, who tries to accommodate the requests of train operators as much as possible, while abiding to safety rules and other operational constraints. Once the timetables are set up, train operators will assign rolling stock and crews to the corresponding tasks.

During real-life operations a train can easily deviate from its nominal timetable: extra time might be needed at a station to board and alight passengers, weather conditions might force the driver to slow down in certain parts of the route, etc. These are examples of *primary* delays. A delayed train, in fact, could interfere with the operations of other trains, in turn delaying them (*secondary* delays) and many delays can end up knocking on from one train to another.

As already mentioned in Section 1, in this work we consider disturbances and disruptions (introduced in Section 1) under a unified umbrella. A detailed list of the conflicts we consider is given in Section 4.3. The corrective actions that our algorithm will suggest are limited to *retiming, respeeding*, and *rerouting* trains, collectively named *rescheduling*. Retiming consists in changing the durations of train stops at stations. Respeeding changes the times trains enter and leave different parts of the network (i.e., changing their speed). Finally, rerouting consists in assigning a train a new path in the network.

Several criteria can be considered when rescheduling a set of trains. For example, we may want to minimise the deviance from the nominal timetable, or the total delay, or the number of broken connections, etc. In our work, we present a general way of modelling events in the network, that is able to take into account all of these criteria (and many more).

3. Literature review

Conflict resolution in train applications, often known as the *train dispatching problem* (see, e.g., Meng and Zhou, 2014), is widely studied in the literature, and research contributions can be classified in several ways.

A first possible subdivision may take into account the level of detail used in modelling the physical resources composing the train network. In this respect, the main distinction was usually between *microscopic* and *macroscopic* modelling approaches. A microscopic approach would represent every element of the rail infrastructure in detail (individual tracks, platforms, etc.). In such a model every network element can be assigned to only one train at a time, thus leading to *explicit* capacity requirements on the resources. A typical macroscopic model, on the other hand, would disregard any fine-grained segmentation of the tracks, thus leading to *cumulative* capacity requirements, since each network element could represent several physical resources. In the literature, such models are also known, respectively, as *single-track* and *N-track* models (see, e.g., Törnquist and Persson, 2007a). This distinction, however, is often blurry, and several authors adopted a mixed approach, by considering so-called *mesoscopic* models, in which the modelling detail is not specified a priori. Here, network elements can represent either low level infrastructure, such as specific tracks or platforms, or aggregate one, such as entire stations or *N*-track segments.

Another widely used subdivision takes into account the type of conflict resolution actions available to the decision makers. These include the application of *retiming, reordering, retracking*, and *rerouting* of trains (Meng and Zhou, 2014). Such actions involve, respectively: the adjustment of speeds and stopping times; modifying the order in which trains occupy platforms or track segments; small and large changes in the path followed by trains in the network.

In their recent survey, Cacchiani et al. (2014) also adopted a classification scheme which mainly takes into account the type of conflict to be managed by the model. More precisely, the authors distinguished between *disturbances* and *disruptions* and analysed the literature classifying models and solution approaches based on this viewpoint. The reader is also referred to Törnquist and Persson (2007a) and Meng and Zhou (2014) for additional literature analyses and classification. Other classification schemes proposed in recent surveys mainly focus on the solution methodology adopted (see Fang et al., 2015) or on dynamic and stochastic components related to on-line rescheduling (see Corman and Meng, 2015). Finally, we direct the interested reader to the recent book of Hansen and Pachl (2014) for a comprehensive analysis of many aspects of railway timetabling and operations, including train rescheduling.

Many works which employ a more microscopic approach revolve around the concept of *alternative graph*, introduced by Mascis and Pacciarelli (2002) for the no-wait job shop scheduling. The problem of assigning a train to a track segment for a certain period of time, in fact, can be seen as a job shop scheduling problem where track segment are machines and the assignment of a train to a segment is an operation. Additional constraints, such as set-up times and no-wait constraints, are used to model specific characteristics of the problem. The alternative graph formulation was widely used to develop solution approaches to various rescheduling problems (see, e.g., D'Ariano et al., 2007) such as the ROMA tool (see, e.g., D'Ariano et al., 2008a; 2008b; Corman et al., 2009; 2010a; 2010b; 2011; 2012; D'Ariano and Pranzo, 2009). Corman et al. (2016) integrates fast heuristics for this model, which provide a primal solution, with a network-flow relaxation, providing a dual bound.

Other approaches, which use alternative solution paradigms, have also been explored. Rodriguez (2007) solved conflicts using constraint programming techniques and using the job shop model with additional constraints. Meng and Zhou (2011) propose a stochastic programming model is used to reschedule trains on a single-track line, so that the new schedule is robust. Pellegrini et al. (2014) solve a real-time traffic management problem using a pure Mixed-Integer Programming (MIP) model which represents a small section of a railway network with fine granularity. Samà et al. (2016) use an ant-colony optimisation metaheuristic to select the best routing alternative for each train in a real-time setting. A simulation-based approach for train dispatching was proposed by Li et al. (2008). Mu and Dessouky (2011) employs fuzzy optimisation techniques to reschedule trains after a low-probability disruption occurs. Finally,

Several authors tried to bridge the gap between fine-grained and more aggregate representations by using different techniques. For example, Lamorgese and Mannino (2013); 2015) propose an iterative macro- and microscopic approach, in which the *line traffic control* problem takes care of the macroscopic constraints (trains meeting at stations, stations' capacities respected) and acts as a master problem. The *station traffic control* considers instead detailed constraints at the station level and acts as a subproblem to generate cuts for the master problem, in a way analogous to Benders decomposition. Other mesoscopic approaches have been based on MIP formulations: Törnquist and Persson (2007b) used an exact model for rescheduling on *N*-track networks; Törnquist (2012) used a MIP-based greedy heuristic starting from the same model; this model was further extended by Acuna-Agost et al. (2011), who also consider intermediate stops and bidirectional tracks.

While minimising the total delay is a sensible choice, in recent years the focus of rescheduling techniques has been shifting towards a more passenger-oriented point of view, which aims to minimise the travellers' delay. In this spirit, Schöbel (2007) solved the delay management problem, consisting in deciding which connections between trains should be maintained, even when this would mean to introduce some delay on certain trains that would have to wait for others. The work has been expanded in Schöbel (2009), Schachtebeck and Schöbel (2010), and Dollevoet et al. (2014), while (Kanai et al., 2011) propose a combined optimisation/simulation algorithm that allows to track additional performance indicators other than total passenger delay. On the other hand, concerning the scheduling of freight trains, Mu and Dessouky (2011) recently proposed effective heuristic approaches based on decomposition.

4. Problem description

Given a description of the current state of the network, the goal of our algorithm is to produce a new timetable for the trains, keeping in mind what was the original, nominal timetable published to the users. There are, therefore, three main objects that we need to model to provide input data to the algorithm: the first is a description of the *physical network*; the second is the *nominal timetable*, the third is the current status of the trains in the network (called the *forecast timetable*).

4.1. Network and timetables

The main tool we use to represent the train network is the *network (di)graph* $G_N = (V, A)$. Nodes in V represent resources. What a resource is can vary greatly and depends mostly on the level of detail we want to achieve when modelling the train network. At a microscopic level, a resource could be a single section of track between two signals, a platform at a station, a junction between tracks, etc. On a macroscopic level, it could be a whole station, or a set of parallel tracks between two stations, etc.

There are, of course, trade-offs between macroscopic and microscopic representations. While the latter will produce a larger graph, using the former will lead us to lose some information. For example, when we represent a set of parallel tracks as a single resource, we can't guarantee that a feasible assignment of trains to the tracks always exists.

Talking about resources rather than more specific railway elements, however, allows us to generalise many aspects of railway networks and even to mix micro- and macroscopic representations in the same graph. This is useful, for example, when the central part of the network is particularly congested and needs a higher level of detail, while peripheral parts are less loaded and can be modelled at a lower resolution.

For example, Fig. 1a shows a macroscopic modelling of a station S and two set of tracks L and R. Fig. 1b shows the same station and tracks at a microscopic level: the station has been substituted by three platforms and the generic set of tracks have been replaced by a node for each physical track. Furthermore, connecting tracks have been introduced, to model the connections between the platforms and the tracks. Notice that a solution that was feasible in Fig. 1a might not be feasible in Fig. 1b. For example, a train leaving P_1 to reach R_2 and a train leaving P_2 to reach R_1 can't depart at the same time, as they would violate the capacity of S_{R2} (which is 1), but we wouldn't have been able to rule out this solution just by looking at Fig. 1a and the capacities of the aggregate nodes.

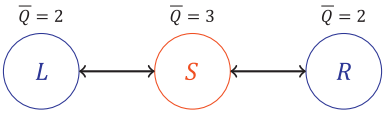
We identified certain properties that apply to all resources, no matter what parts of the physical network they represent:

- Every node $v \in V$ has an ideal (or *soft*) capacity $Q_v \in \mathbb{N}$ and a *hard* capacity $\overline{Q}_v \in \mathbb{N}$. The capacity of a resource indicates the number of trains that can occupy it at the same time. While the soft capacity can be violated (by possibly paying a certain penalty) the hard capacity cannot be violated under any circumstances. The relation $Q_v \leq \overline{Q}_v$ holds.
- Every node $v \in V$ also has an associated boolean parameter, $\omega_v \in \{0, 1\}$, that indicates whether overtaking and crossing between trains can happen at the node.

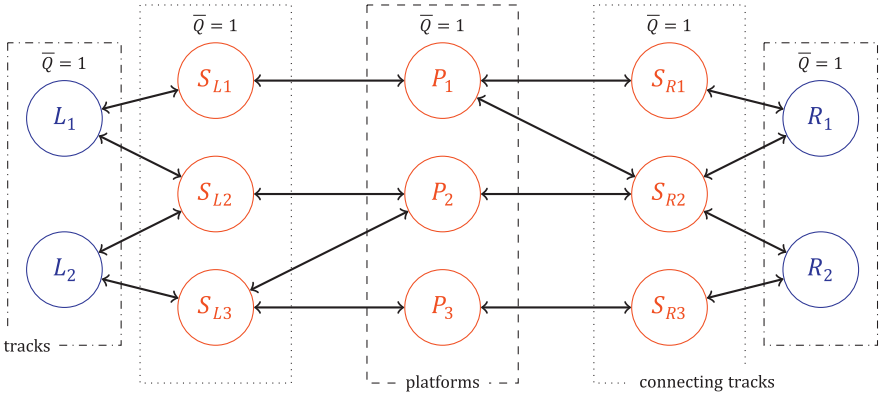
The arcs in set A represent the possibility for trains to move from one node to another. Arcs also have capacities, indicating the number of trains that can simultaneously transit from the source to the destination node of the arc: we indicate the capacity of $a \in A$ with $\overline{Q}_a \in \mathbb{N}$. This quantity is considered as a hard capacity.

The other main actors of a train network are, naturally, trains. Let I be the set of trains and consider the following properties that link together resources and trains:

- Given a train $i \in I$ and a resource $v \in V$, we give the minimum and maximum *travel times*, i.e., the minimum and maximum times that i is allowed to occupy v. We denote these values with $m_{i,v}$ and $M_{i,v}$ respectively. The physical meaning of these quantities can vary depending on what the resource models. In case of a section of track, $m_{i,v}$ is given by the



(a) Macroscopic representation of station S with tracks L on its left and R on its right.



(b) Microscopic representation of the same station as in Figure 1a, with all platforms and physical tracks modelled explicitly.

Fig. 1. Differences between the micro- and macroscopic representations of a station. $\overline{Q}$ represents the capacity of a resource.

length of the track and the maximum speed that the train can achieve on that track. On the other hand, in case of a platform, $m_{i,v}$ is the dwelling time.
- Given a resource $v \in V$, we denote with h_v the minimum *headway* at v, i.e., the time that must elapse between two trains occupying the resource.

The nominal timetable describes the ideal operational status of the network. Each train $i \in I$ has a predefined path in the network, denoted as $p_i = (v_1, v_2, \ldots, v_{k_i})$, which is simply a sequence of resources to be visited: $v_1, \ldots, v_{k_i} \in V$.

For each node in the path of train i, the nominal timetable also provides the times at which the train is supposed to enter and leave the node. These times are denoted as $\theta^{\text{in}}_{i,v_j}$ and $\theta^{\text{out}}_{i,v_j}$ respectively.

The current train plan describes the network as it is at the present moment — and as it is forecast to be in the future, given the information available. For this reason, such a plan is also called the *forecast timetable*. In an ideal scenario, the forecast timetable is always equal to the nominal one. In practice, when a disturbance or a disruption occurs, the forecast diverges from the nominal timetable.

The forecast timetable has a formal structure which is similar to that of the nominal timetable: it gives a sequence of nodes that each train must visit, together with the expected in- and out-times. Since both timetables describe the (ideal and real, respectively) situation of the network before the dispatcher takes any decision regarding rerouting, the train paths must be the same in both.

The only new parameters associated with the forecast are, therefore, the in- and out-times. To account for the uncertainty that comes with the real-time situation of the network, we actually give pairs of minimum and maximum possible in- and out-times. These values should be considered as hard values, i.e., the train cannot possibly enter a node before the minimum in-time or after the maximum in-time.

The minimum in- and out-times are denoted, respectively, as t^{in}_{i,v_j} and t^{out}_{i,v_j}, for $j = 1, \ldots, k$. The maximum in- and out-times are T^{in}_{i,v_j} and T^{out}_{i,v_j}.

As we mentioned in Section 2, rescheduling a train can involve rerouting it. This means that the dispatcher is allowed to change the path of the train in the network. In our model, we assume that it is only possible to choose detours from a predefined set available for each train. The set of detours associated with train i is D_i.

A detour is nothing more than a path in the network, so an element $d \in D_i$ contains a sequence of nodes: $d = (v_1, v_2, \ldots, v_k)$. We only require that both the first and the last node of the detour are also part of the original train path p_i. Furthermore, similarly to what we have seen for the current train plan, maximum and minimum in- and out-times are given for each node v_j of the detour. These are denoted as t^{in}_{i,d,v_j} (minimum in-time), T^{in}_{i,d,v_j} (maximum in-time), t^{out}_{i,d,v_j} (minimum out-time), and T^{out}_{i,d,v_j} (maximum out-time).

4.2. Time-space graph

The network graph $G_N = (V, A)$ introduced in Section 4.1 does not explicitly model the time component. In this subsection, we present a time-space graph and we construct it starting from G_N and augmenting the number of nodes to take into account time and entry/exit points of nodes of V. Time-expanded graphs have already been used to model railway networks, e.g., in Caprara et al. (2002) and Cacchiani et al. (2010). The *time-space (di)graph* of a train $i \in I$ is denoted as $G^i_{TS} = (V^i_{TS}, A^i_{TS})$ and is obtained in the following way.

For every entry and exit point of every node $v \in V$, a node is added to V^i_{TS}. The definition of entry and exit point is strongly dependent on the physical resource modelled by v. For example, if v represents a set of parallel tracks, there will be one entry and one exit point for each track; if v models a station, there would be an entry and one exit point for every track running through the station. In general, the number of entry and exit point does not need to match (e.g., a station could have more tracks one side than the other). The names *entry* and *exit* are only used to distinguish two physical locations on the resource, but since trains can generally run on a resource in both directions, a specific train could actually enter the resource from one of its exit points and leave it from one of the entry points.

We then need to model time into the graph. In order to do this, we first have to decide a reasonable time horizon and a time discretisation. In practical applications, these values could be provided to the model by the upstream conflict detection system. Notice, though, that the flexibility bundled with our model allows us to use different time discretisations in different parts of the time-space graph: some resources or some time intervals can be modelled with a more precise time discretisation than others. For example, it is possible to have a denser time discretisation for peak times and a sparser one for low-congestion times (e.g., at night). A denser discretisation might also be necessary for short tracks, where the travelling time could be shorter than the standard time interval. Once the time discretisation and the time horizon have been fixed, each node gets one further copy per time instant.

Finally, two dummy nodes σ^i_{src} and σ^i_{snk} are added to each graph G^i_{TS}. They represent, respectively, a source and sink node used as the start and end point of the train's path in the graph.

Arcs are created between pairs of nodes $(w_1, w_2) \in V^i_{TS}$ and they are divided in three types. The first type links nodes which represent entry and exit point relative to the same resource $v \in V$. Such an arc would model the travelling of a train along the resource modelled by v, when the difference in time instants represents a feasible travelling time for train i.

The second type links nodes which represent entry and exit points of adjacent resources, that is of nodes $v_1, v_2 \in V$ such that $(v_1, v_2) \in A$. Such an arc would model a train that leaves a resource and (instantaneously) reaches a new one.

Finally, the third type links the source and the sink to the other nodes. Let w^i_s and w^i_e be the entry points that train i has to use to access and leave, respectively, its start and end resources. We then add to the arc set A^i_{TS} a: (a) arcs from σ^i_{src} to nodes of the form (w^i_s, t), where t is a time instant; (b) arcs from nodes of the form (w^i_e, t) to σ^i_{snk}, where t is a time instant; (c) arcs from nodes of the form (w, T) to σ^i_{snk}, where w is any entry or exit point, and T is the last time instant of the time horizon, used to represent a train that could not reach its destination within the time horizon considered.

We list the three type of arcs separately and let $A^i_{TS} = A^{i,1}_{TS} \cup A^{i,2}_{TS} \cup A^{i,3}_{TS}$, where the three sets contain, respectively, arcs of the three types listed above.

Fig. 2 shows a portion of a time-expanded graph. Nodes w^L_1 and w^R_1 are the left and right extreme points of $v_1 \in V$, while nodes w^L_2 and w^R_2 are the left and right extreme points of $v_2 \in V$. The arcs between w^L_1 and w^R_1 represent the traversal of resource v_1 and, analogously, the arcs between w^L_2 and w^R_2 represent the traversal of resource v_2. Different arcs having the same source node model the different travelling times associated to different speeds. The vertical arcs between w^R_1 and w^L_2 represent the possibility of moving from v_1 to v_2.

The arcs in $A^{i,1}_{TS}$ can be mapped back to the resources and the time intervals they represent in the following way. For each arc $a \in A^{i,1}_{TS}$, let $\rho(a) \in V$ be the underlying resource modelled by the arc; analogously, let $\tilde{\rho}(a) \in V \times \{-1, +1\}$ be the directed underlying resource, used to distinguish the direction in which the resource is being traversed; let $l(a)$ be the length of the associated resource $\rho(a)$. Let also $\lambda_s(a)$ and $\lambda_e(a)$ be the start and end time of arc a, i.e. the times when the train (respectively) occupies and frees the resource.

4.3. Constraints

As defined in Section 1, conflicts are those situations that either can't physically happen or that would compromise the safety of operations, and their resolution plays the same role as satisfying a constraint in a Mixed Integer Programme (MIP). In order to give a more precise description of the constraints presented in the rest of this section, we give some mathematical formulation in which we use the notation $x^i_a \in \{0, 1\}$ as a variable in a Mixed Integer Programme, having value 1 iff the arc $a \in A^i_{TS}$ is part of the path of train i.

Preliminary experiments with solving a compact MIP formulation of real-life instances with a commercial solver have shown that model generation alone can take several minutes, and solving the root node requires more than a day. For this reason, we do not include a complete MIP model for the problem we are presenting. Rather, the notation x^i_a should be seen as a way to describe precisely the constraints taken into account by our algorithm, and how they are reflected on the time-space graph.

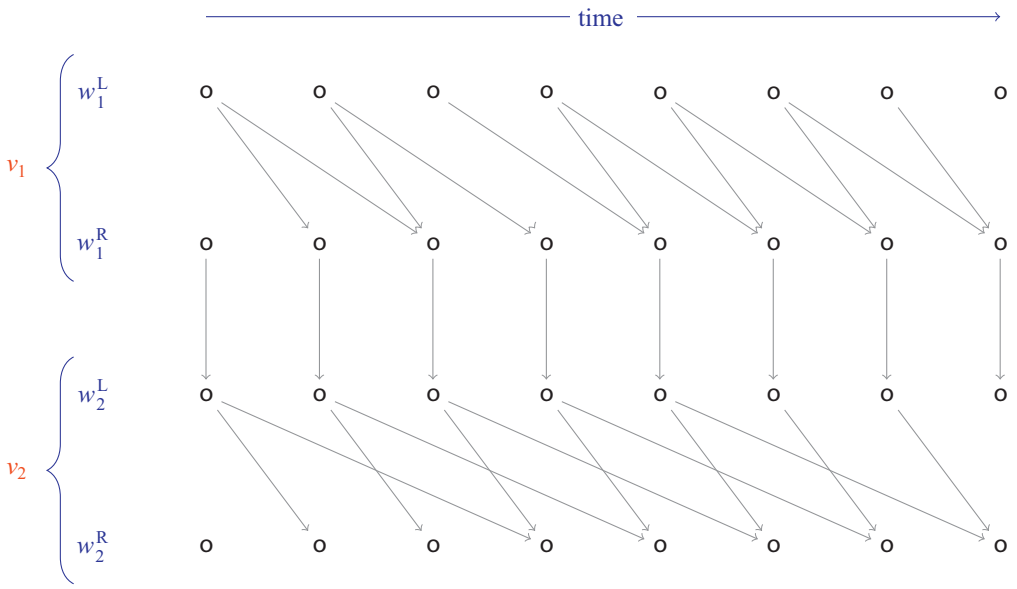


Fig. 2. Example of a portion of time expanded graph.

Notice, first of all, that a train schedule can be modelled as a path in G^i_{TS}, starting in σ^i_{src} and ending in σ^i_{snk} and abiding to the usual flow conservation constraint. Formally, this means that:

$$\sum_{a \in A^{i,+}_{TS}(\sigma^i_{src})} x^i_a = 1 \tag{1}$$

$$\sum_{a \in A^{i,-}_{TS}(\sigma^i_{snk})} x^i_a = 1 \tag{2}$$

$$\sum_{a \in A^{i,-}_{TS}(w)} x^i_a = \sum_{a \in A^{i,+}_{TS}(w)} x^i_a \qquad \forall w \in V^i_{TS} \setminus \{\sigma^i_{src}, \sigma^i_{snk}\} \tag{3}$$

where $A^{i,+}_{TS}(w)$ (resp. $A^{i,-}_{TS}(w)$) is the set of all arcs outgoing from (resp. incoming to) node $w \in V^i_{TS}$.

In this work we deal with the rescheduling of the trains once the conflicts have already been detected and reported, so we will assume that the complete list of conflicts is available together with the current train plan. In other words, the conflict detection system works upstream of our system. This assumption is not restrictive, as a simple linear-time algorithm over the current train plan is able to produce the complete list of conflicts.

The presence of conflicts could be formally detected by checking for violations in (hard) constraints involving the variables x^i_a. As we will see in Section 4.4, we want to penalise the violation of certain soft constraints. In a MIP model, for example, a hard capacity limit can be enforced via a constraint, whereas the extent of the violation of a soft capacity limit can be penalised, by introducing an auxiliary variable that plays the role of the slack variable relative to the constraint.

4.3.1. Illegal crossing and overtake

An illegal crossing (overtake) describes a situation when a train would cross (overtake) with another one, on a resource v where this is illegal, i.e. $\omega_v = 0$. An example of overtaking is described in Fig. 3.

An overtake corresponds to the violation of the following constraints: for each train i, each resource v where overtaking is forbidden, and each arc $a \in A^{i,1}_{TS}$ such that $\rho(a) = v$:

$$\sum_{\substack{j \in I \\ j \neq i}} \sum_{\substack{a' \in A^{j,1}_{TS} \\ \tilde{\rho}(a') = \tilde{\rho}(a), \\ \lambda_s(a') > \lambda_s(a), \\ \lambda_e(a') \leq \lambda_e(a)}} x^j_{a'} + x^i_a \leq 1 \tag{4}$$

Constraint (4) states that train j overtakes train i on resource v if j arrives in v after i, but leaves v before i, and both trains travel in the same direction.

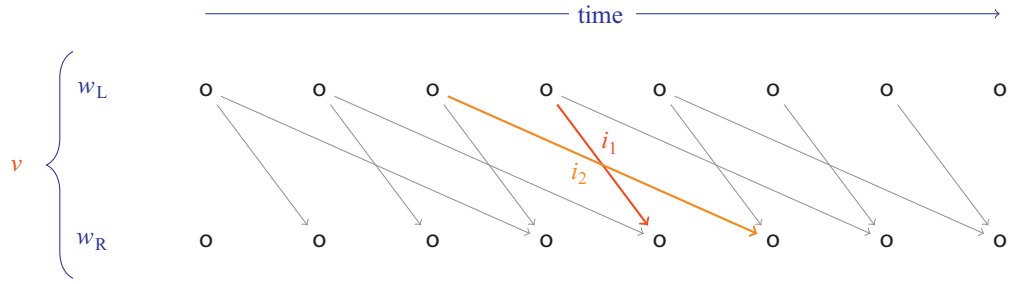


Fig. 3. Train i_1 overtaking train i_2 on resource v. Notice how this situation corresponds to crossing arcs in the time-space graph.

Analogously, corresponds to the violation of the following constraints: for each train i, each resource v on which crossing is forbidden, and each arc $a \in A^{i,1}_{TS}$ such that $\rho(a) = v$:

$$\sum_{\substack{j \in I \\ j \neq i}} \sum_{\substack{a' \in A^{j,1}_{TS} \\ \rho(a') = \rho(a), \\ \tilde{\rho}(a') \neq \tilde{\rho}(a), \\ \lambda_e(a) \geq \lambda_s(a')}} x^j_{a'} + x^i_a \leq 1 \tag{5}$$

Constraint (5) states that train j crosses train i on resource v if j arrives in v before i has left, and the two trains travel in opposite directions.

4.3.2. Capacity violation

A capacity violation occurs when the number of trains simultaneously occupying a resource is greater of the hard capacity of the resource. Such a conflict corresponds to a violated inequality of the following type:

$$\sum_{i \in I} \sum_{\substack{a \in A^i_{TS} \\ \rho(a) = v, \\ \lambda_s(a) \leq t, \\ \lambda_e(a) \geq t}} x^i_a \leq \overline{Q}_v \tag{6}$$

for each resource v and each time instant t, where we remind that $\overline{Q}_v$ is the hard capacity of resource v.

4.3.3. Headway violation

Such a conflict occurs when a train occupies a resource that has been occupied by another train, and not enough time has elapsed between the first train leaving the resource and the second one entering it. For each train $i \in I$ and each arc $a \in A^{i,1}_{TS}$ corresponding to a resource $\rho(a)$ on which crossing and overtaking is forbidden ($\omega_{\rho(a)} = 0$, as otherwise the headway must not be respected), a headway conflict corresponds to a violated constraint:

$$\sum_{\substack{j \in I \\ j \neq i}} \sum_{\substack{a' \in A^{j,1}_{TS} \\ \rho(a') = \rho(a) \\ \lambda_e(a') \geq \lambda_s(a) - h_{r(a)} \\ \lambda_s(a') \leq \lambda_e(a) + h_{r(a)}}} x^j_{a'} + x^i_a \leq 1 \tag{7}$$

4.3.4. Time dependencies

Avoiding the conflicts described in the previous subsection is usually enough to come up with a new plan that allows safe operations and limits the deviations from the nominal timetable. Unfortunately, this is not always enough to provide a holistic, good solution.

Consider, for example, a passenger on a delayed train that risks missing his connection. From his point of view, a solution that also delays his next train (to "wait for him") is preferable to a solution that does not. But from a train operator's point of view, a solution that does not delay the second train may be considered better, since no delay is better than some delay.

Then, if the train operator's *service intention* is, for example, "moving passengers from A to B" (even at the cost of increasing the overall delay, to some extent), this should be taken into account by the rescheduling algorithm.

In order to take into account service intentions, we introduce the concept of *time dependencies* (Caimi et al., 2011). A time dependency is a relationship of precedence between two events happening in the network. For example, a time dependency could mandate that the event "train i_1 leaves node v" can only happen a certain time after the event "train i_2 arrives at node v"; intuitively, we would say that train i_1 needs to wait for train i_2 at node v.

A service intention can merely suggest a precedence between events, as in the case of passenger connections: it is preferable that the connection is kept, but this "promise" can be broken in order to improve the overall quality of service. On the other hand, the precedence can also be mandatory, as in the case of two trains that share crew or rolling stock.

Let F be the set of time dependencies. We associate to each element $f \in F$ the following values:

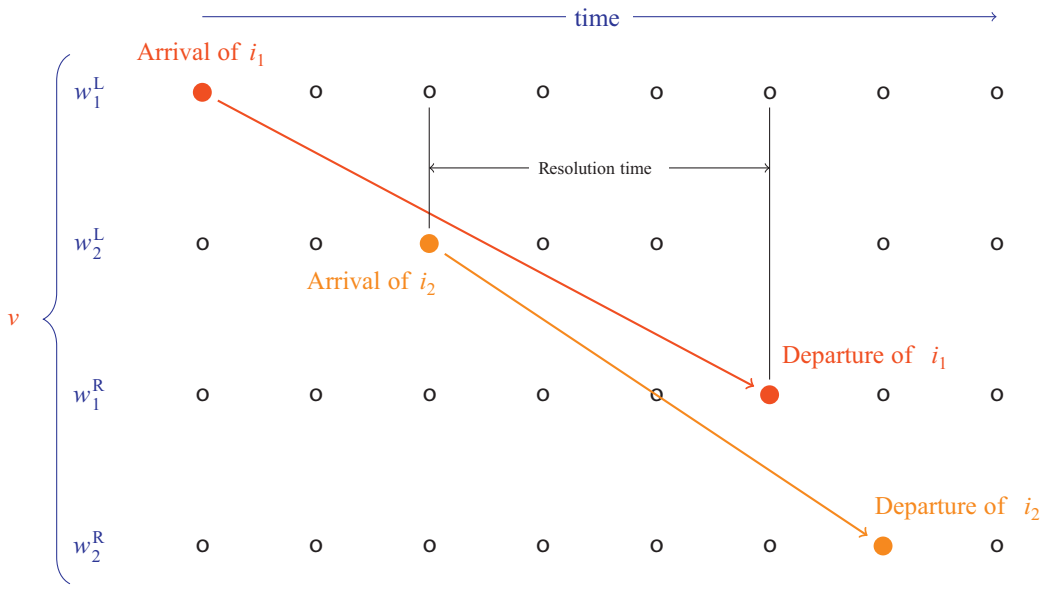


Fig. 4. Resolution time of a time dependency between the arrival of train i_2 and the departure of train i_1 at a station.

- The two trains $i_{f,1}$, $i_{f,2}$ involved in the dependency.
- The two resources $v_{f,1}$, $v_{f,2}$ at which the linked events need to take place, respectively.
- Two parameters $\varepsilon_{f,1}$, $\varepsilon_{f,2} \in \{0, 1\}$ that take value 0 if the corresponding event is an arrival or value 1 if it is a departure.
- A parameter $\eta_f \in \{0, 1\}$ that takes value 1 iff the dependency is mandatory.
- The minimum and maximum resolution time, φ_f and Φ_f. The dependency is considered satisfied if the two events take place at least φ_f and at most Φ_f time units apart.
- For non-mandatory (also called *logical*) time dependencies, we give a maximum waiting time w_f. The dependency must be satisfied if it is possible to do so by introducing at most w_f time units of delay. If this cannot be done, the dependency can either be satisfied or not. For example, in the case of a connection between two trains, we might require that a train waits for the other at most w_f time units. If the required wait is greater, the train can decide to break the connection.

Fig. 4 shows an example in which train i_1 stays at station v for three time intervals after the arrival of train i_2, presumably to *wait* for passengers travelling on i_2. The resources w_k^L, w_k^R represent entry and exit points of two platform at the station $(k = 1, 2)$.

The violation of a mandatory time dependency $f \in F$ can be detected as follows. Assume wlog that the event relative to train $i_{f,1}$ needs to take place before the event relative to train $i_{f,2}$. For $k = 1, 2$ and $a \in A_{TS}^{i_{f,k},1}$ such that $\rho(a) = v_{f,k}$, consider the parameter:

$$\delta_{f,k}(a) = \begin{cases} \lambda_s(a) & \text{if } \varepsilon_{f,k} = 1 \\ \lambda_e(a) & \text{if } \varepsilon_{f,k} = 0 \end{cases}$$

We can then formulate the corresponding constraint:

$$\varphi_f \leq \sum_{\substack{a \in A_{TS}^{i_{f,2},1} \\ \rho(a) = v_{f,2}}} x_a^i \delta_{f,2}(a) - \sum_{\substack{a \in A_{TS}^{i_{f,1},1} \\ \rho(a) = v_{f,1}}} x_a^i \delta_{f,1}(a) \leq \Phi_f \tag{8}$$

4.3.5. Split and merge

Similar to time dependencies are split and merge operations. These are events in which the trains that enter a node are not the same that leave it. They model real-life operations such as decoupling some cars from a train, so that they can get a new locomotive and proceed to a different destination.

In the most general version, any number of trains can enter a certain node and any number of trains can leave it, so a split/merge is identified by a node $v \in V$ and two sets of trains $I_1, I_2 \subset I$ that represent in-trains and out-trains. The out-trains can leave the node only after a certain amount of time has passed since the last in-train reached it. This time accounts, in practice, for the time necessary to perform any physical coupling and decoupling, or to change crew or rolling stock. Special cases of split/merge operations are:

- *Split*, when one train enters the node, and two or more exit it.

- *Merge*, when more than one train enter the node, and only one exits it.
- *Rename*, when one train enters the node and one train exits it.

In case of split/merge events, the node's capacity is not considered, as it is assumed that it is always feasible for the event to take place in the node specified. Finally, note that these events are mandatory: for example, it is not possible that a train will be detoured around a node in which it has to undergo a split.

In the following we show how a split or merge operation can be modelled in terms of mandatory time dependencies; in this way, the conflict resulting from a missed split or merge can be detected by checking for the violation of the corresponding time dependency constraint. For example, if train i needs to be split into trains i', i'' at resource v, then said resource will be set as the destination of i and the origin of i' and i''. Furthermore, two dependencies will be created:

- $f' \in F$ links trains i and i' and has: $i_{f',1} = i, i_{f',2} = i'$; $v_{f',1} = v_{f',2} = v$; $\varepsilon_{f',1} = 0, \varepsilon_{f',2} = 1$; $\eta_{f'} = 1$; $\varphi_{f'}$ will be the time needed to perform the split operation; $\Phi_{f'}$ will be the maximum time allowed for the split to take place, if any.
- $f'' \in F$, analogously links trains i and i''.

4.3.6. Maximum and minimum entry, exit, and travel times

Maximum and minimum entry, exit, and travel times can be enforced by removing from the graphs G^i_{TS} those nodes that would correspond to an infeasible (resource, time) couple. For example, if a train $i \in I$ cannot enter resource v before time t, all nodes of V^i_{TS} corresponding to resource v at a time $t' < t$ can be removed from the graph. Analogously, if the minimum travel time along a resource v is t, all arcs $a \in A^{i,1}_{\mathrm{TS}}$ such that $\lambda_{\mathrm{e}}(a) - \lambda_{\mathrm{s}}(a) < t$ can be removed.

4.4. Objective function

The objective value can be written as a function of the paths in the time-space graphs, and therefore of $\vec{x} = (x^i_a)$, in the following way:

$$f(\vec{x}) = f_1(\vec{x}) + f_2(\vec{x}) + f_3(\vec{x}) + f_4(\vec{x}) \tag{9}$$

The four components correspond to delays, logical dependency breaking, soft capacity violations, and the use of detours. Each of these components represents a sum of penalties that quantify how undesirable it is to incur in the corresponding violations. The penalty, therefore, is not only limited to represent the economical disadvantage of taking a particular decision (e.g., increased energy consumption) but can also represent intangible values, such as customer satisfaction. In the following, we analyse these four components.

4.4.1. Delays

In the nominal timetable, we associated to each train i and each resource v_j in the train's path, an ideal in-time $\theta^{\mathrm{in}}_{i,v_j}$ and an ideal out-time $\theta^{\mathrm{out}}_{i,v_j}$. Any deviation from these times can be penalised, by considering two piecewise-linear functions that respectively assign a cost to delays in arriving at and departing from the resource. These penalty functions are denoted as $\pi^{\mathrm{ind}}_{i,v_j}(\cdot)$ and $\pi^{\mathrm{outd}}_{i,v_j}(\cdot)$.

Notice that this general definition allows us to assign different penalty profiles to different resources: for example, if some resource is considered critical for a train, we can assign a higher penalty to delays at that resource. The function can also operate on negative delays, allowing us to penalise trains that arrive at a node with excessive advance. Finally, further flexibility is bundled in the piecewise-linear nature of the function: for example, we might want to have a penalty that grows linearly with the delay up to a certain point, after which a big flat penalty is assigned, as any further delay does not worsen the situation any more. This could be achieved with a penalty profile such as that in Fig. 5.

4.4.2. Dependency breaking

As we mentioned in Section 4.3.4, logical dependencies are not mandatory and therefore we can decide to break them. In our implementation, when such a dependency $f \in F$ is broken, we pay a penalty π^{dep}_f. (Notice that, in principle, a piecewise linear function could be used, as done for delays.)

4.4.3. Capacity violations

When we violate the soft capacity Q_v of a resource v, we pay a penalty π^{cap}_v. In our implementation, this penalty remains the same no matter how big the capacity violation is. (Notice that, in principle, a piecewise linear function could be used in this case too.)

4.4.4. Detours

Finally, a fixed penalty π^{det}_d is paid when a train i is re-routed along a detour $d \in D_i$. This penalty takes into account all the costs (economical or otherwise) incurred because of the rerouting. Dependency breaking and capacity violations are calculated separately when a detour is taken. If some node of the detour can be naturally mapped to nodes of the original path, the delay penalty can also be calculated. For example, if the detour consists of a platform change at a station, we can naturally assign to the new platform the in- and out-times at the old platform.

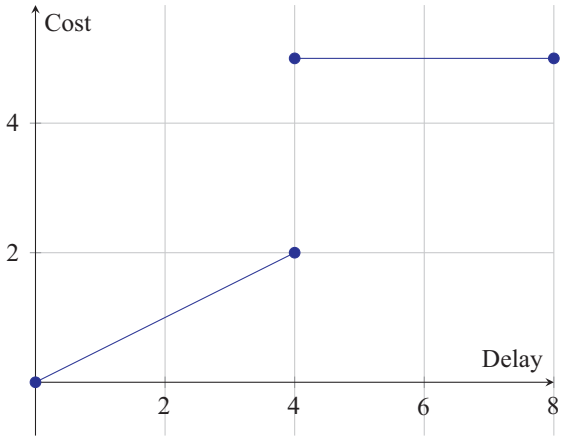


Fig. 5. Example of a penalty profile for the arrival of a train at a certain resource, with a "jump" in cost if the delay is greater than 4 time units.

The penalty π_d^{det} should be considered as a fixed cost incurred by the mere fact of having rerouted the train. It can include both *real* and *virtual* costs. For example, if the detour consist of a path longer than the original one, there would be increased energy costs. But if the detour also excludes a station, there would be a much increased passenger dissatisfaction. Therefore, the magnitude of the penalty depends on the type of detour: a platform change will have a small associated penalty, while a major change in the train's path will be associated with a bigger penalty.

4.4.5. Other terms

Even though in the present work we only consider the four components discussed above, further terms can be easily introduced in the objective function to take into account other indicators. For example:

- Number of modifications with respect to the nominal timetable, eventually weighted differently depending on the nodes or trains involved. This could be done to make sure that the new timetable does not disrupt too much the current operations (i.e., changes too many train paths) just to save a few seconds of overall delay.
- Increased travel time on resources, eventually involving a piecewise-linear penalty profile. This term would help avoiding unnecessary stops or excessive brake-accelerate cycles, for increased passenger comfort and reduced energy consumption.

5. Solution algorithm

A solution to the TRP is a collection of paths, one for each train, in the time-space graph G_{TS}. The solution algorithm must be able to modify the forecast timetable in order to solve the conflicts, and to compute the new objective function. Key requirements for this algorithm, deriving from its real-time nature, are:

- It must produce solutions of high quality in very short computational times (a few seconds). This is due to the fact that the algorithm is used on-line by dispatcher, who needs reasonable advice in few seconds, to guarantee the safety of operations in the network. For this reason, we focussed on a heuristic approach.
- If the algorithm is not able to find a conflict-free schedule, it has to give the dispatcher a schedule with the smallest possible number of remaining conflicts. This means that solving more conflicts needs to always have priority over other factors. In order to accomplish this, we established an implicit hierarchy through our objective function. A very high penalty is assigned to each unresolved conflict, so that a solution with fewer conflicts will always prevail on one which has more, while the "standard" objective function is used to decide the best one between two solutions with the same number of remaining conflicts. Therefore, the objective function presented in (9) is used as a part of the overall objective function: $\tilde{f}(\vec{x}) = P \cdot N_{\text{RC}}(\vec{x}) + f(\vec{x})$, where P is the large penalty to pay for each conflict, and N_{RC} gives the number of remaining conflicts in the solution.
- The algorithm should allow for a high degree of parallelisation, allowing to concurrently produce multiple rescheduled timetables that will be stored in a solution pool, from which the best one will be selected. This allows the algorithm to employ and evaluate different strategies in situations where none of them is clearly superior to the others, thereby focussing on different key aspects of the problem.

With these requirements in mind, we now give a general description of the algorithm which broadly falls into the category of iterated greedy algorithms (see, e.g., Ruiz and Stützle, 2007). After an initialisation phase in which trains are ranked according to some criterion, the algorithm iterates among two main phases, until a termination condition is met. The two phases are:

1. **Construction**: a new timetable is obtained by rescheduling the trains one by one, according to their ranking.

2. **Shaking**: the train ranking is perturbed following a set of rules.

In our case the termination criterion is a hard time limit, with early termination if the solutions didn't improve over a certain number of iterations. The next subsections will describe each phase in detail. Furthermore, in order to speed up the computational time of the construction phase, we employed a sparsification of the graph G_{TS}. This is a technique used to remove some edges and vertices from the time-space graph. Its use is justified by the fact that, by choosing a fine time discretisation, we might create a great number of edges, many of which can be removed without strongly impacting the quality of the train plans.

Notice that, for the algorithm initialisation, for the shaking phase, and for sparsification we propose several possible alternatives. Therefore, an instance of our algorithm is completely defined once we specify which initial sorting, shaking policy, and sparsification method is used. An extensive experimental testing of the proposed alternatives, described in Section 6.1, will help determining well-performing combinations of the algorithm's components.

5.1. Initial sorting

Since the algorithm constructs a schedule for one train at a time (Step 1), the order in which the trains are considered is clearly important, as trains scheduled later will be constrained by those scheduled earlier. Since there is no "natural" order of trains, we used various sorting criteria:

- **Random**: the trains are randomly sorted.
- **Congestion**: the trains are sorted according to the number of conflicts in their forecast timetable, putting trains with more conflicts first. The rationale behind this choice is that a train that generates a lot of conflicts is harder to schedule, and therefore should be scheduled earlier. Using the notation introduced earlier, if $\bar{x}^*$ is the assignment of variables corresponding with the forecast timetable, then we sort the trains by decreasing value of $N^i_{RC}(\bar{x}^*)$, where $N^i_{RC}(\bar{x}^*)$ is the number of conflicts in train i's schedule, and $N_{RC}(\bar{x}^*) = \sum_{i \in I} N^i_{RC}(\bar{x}^*)$.
- **Length**: the trains are sorted according to the number of nodes in their original path in decreasing order. The longer the path, in fact, the higher the chances that a conflict will be present at some node. The trains are therefore sorted in decreasing order of the size of their set $\{a \in A^i \ : \ \bar{x}^* = 1\}$.
- **Conflict time**: the trains are sorted according to the time instant of the earliest conflict in their forecast timetable. This is because an early conflict can impact the overall network status at a much later time. In other words, there is more freedom when fixing conflicts happening earlier, and we want to fully use this freedom.
- **Speed**: the trains with highest average speed are scheduled earlier. This strategy is based on the observation that faster trains have schedules that are more sensitive to variations. The average speed of a train i is given by:

$$\frac{\sum_{a \in A^i_{TS},\, \bar{x}^*_a = 1} \frac{l(a)}{\lambda_e(a) - \lambda_s(a)}}{\left| \left\{ a \in A^i_{TS} \ : \ \lambda_e(a) > \lambda_s(a) \text{ and } \bar{x}^*_a = 1 \right\} \right|}$$

During a preliminary experimental phase, we noticed that using the sorting criteria in reverse order can sometimes lead to better results. For this reason we also considered the criteria **Reverse congestion** and **Reverse speed**.

5.2. Construction

Each train schedule is constructed by solving a shortest-path problem on the time-expanded graph G_{TS}, where the starting node corresponds to the current position of the train and the ending node corresponds to the train's desired position at the end of the time horizon.

Since trains run along fixed routes, with only a few possible detours, and since they have hard constraints on the time at which they can reach and leave certain resources, the graph G_{TS} can be pruned accordingly for each train. Once this is done, the shortest path is constructed with a custom label-setting algorithm. Given a partial path to a certain node $(w, t) \in V_{TS}$, the corresponding label will be $L = ((w', t'), c)$ where the node $(w', t') \in V_{TS}$ is the predecessor of (w, t) in the partial path and $c \in \mathbb{R}$ is the cost of the partial path up to the current node.

Notice that, since trains are scheduled sequentially and the algorithm is ran on a different time-space graph for each train, we cannot run into deadlocks. A deadlock will indeed correspond to an unresolved conflict (usually a capacity violation) and be accordingly heavily penalised in the objective function.

We discuss now two main aspects of this algorithm: the order in which we extend the labels and how we update the cost component. Labels are extended greedily, i.e., starting from the one with the lowest cost component, but with one exception: a label related to a resource w will be extended only after all the labels related to resources $w' \prec w$ have already been extended (independently from the time interval), even if they have a higher cost. The relation $\prec$ is the topological order relation between nodes in the subgraph of G_N induced by the union of the path of the train p_i and all the detours in D_i. To better understand this rule, consider Fig. 6, which describes the topological setup of a network with a main corridor and a possible detour (via F and G). Labels related to resource D, for example, will be extended only after all labels related to resources A, B, C, F and G have been extended, since $A \prec B \prec C \prec D$ and $A \prec F \prec G \prec D$.

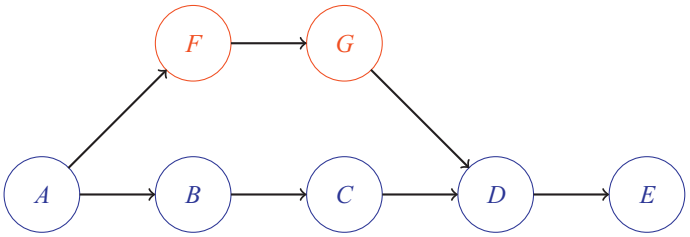


Fig. 6. Topological order between nodes (resources) of a simple network.

The cost component is updated taking into account the various penalties included in the objective function. Some of these penalties, however, depend on the interaction between different trains. As an example, consider a connection between trains i_1 and i_2. If i_1 is scheduled before i_2, when we schedule i_1 will just assume that the connection will be satisfied. If, when scheduling i_2, we realise that the connection is broken, we will add the penalty on the objective function of i_2.

Finally, we need to take special care in case of split/merge operations, as these require the presence of multiple trains on the same resource at the same time. When the partial path of an input train reaches one of these resources, we fix the schedule of the train up to that point, by choosing the lowest cost partial path (we first explore all non-dominated partial paths up to the synchronisation point). We proceed with the construction of the schedule for the output trains, only once all the schedules of the input trains have been fixed up to the considered resource.

5.3. Shaking

In the shaking phase we perturb the ordering of the trains (the *dispatching sequence*) with the aim of finding an ordering which leads, through a new construction phase, to an improving solution. We present two alternative policies, inspired to two well-known metaheuristic algorithms: Reduced Variable Neighbourhood Search (RVNS) and Tabu Search (TS). As mentioned in the beginning of this section, these two alternative policies are experimentally evaluated in Section 6.

The RVNS (see, for example, Hansen et al. (2010)) explores the solution space of the problem by employing a sequence of neighbourhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_K$. A neighbourhood structure defines a way to describe the neighbourhood of any solution x in the solution space.

Starting from a solution x, RVNS generates a new random solution $x' \in \mathcal{N}_1(x)$. If the new solution is not better than the previous one, it goes on to the second neighbourhood structure and generates a random $x' \in \mathcal{N}_2(x)$. The procedure continues until either we run out of neighbourhood structures or the new solution x' is better than the current one x. In the first case, the algorithm terminates; in the second case, the algorithm is restarted using x' as initial solution and going back to using $\mathcal{N}_1$.

The neighbourhood structures are typically such that

$$\mathcal{N}_1(x) \subseteq \mathcal{N}_2(x) \subseteq \ldots \subseteq \mathcal{N}_K(x) \tag{10}$$

for all solutions x. This means that while no improving solution is found, the search space around x is enlarged.

In our case, the neighbourhood structure $\mathcal{N}_k(x)$ consist in considering all dispatching orders that can be obtained from x by performing at most k swaps. From this definition, it follows immediately that (10) is satisfied. The trains to be moved in the new dispatching order are selected at random with a roulette wheel selection procedure where the probability associated to each train is proportional to its contribution to the objective value. The number of positions each train is moved up is again chosen at random, according to a uniform distribution in $[\mu_{\min}, \mu_{\max}]$.

The second policy is inspired to Tabu Search. Starting from a dispatching sequence, we produce a new one analogously to what done with the RVNS policy. We will place in our tabu list the precedence relations between the moved trains. For example, if we transform the sequence $x = (A, B, C, D)$ into $x' = (A, D, B, C)$, we will store the precedence relations (D, B) and (D, C). While these are in the tabu list, the relative order of trains D, B and D, C will not be inverted. If, at the next iteration, train B will be selected to be moved up, then train D will have to move together with B, so not to invert the relation (D, B); the new dispatching sequence will then be (D, B, A, C).

The number of iterations each precedence move is stored in the tabu list depends on three factors:

1. The change in the part of the objective function relative to the moved train;
2. The change in the overall objective function;
3. Whether the new solution improved the incumbent solution.

5.4. Sparsification

As previously mentioned, the sparsification of the graph G_{TS} is used to remove some edges and vertices from the time-space graph, to speed up the computation of shortest paths by the labelling algorithm. Its use is justified by the fact that,

by choosing a fine time discretisation, we might create a great number of edges, many of which can be removed without strongly impacting the quality of the train plans.

As an example, consider a segment of track 5km long and a train that, at full speed, would travel on this segment at 100 km/h. The running time of this train will be of 3 min. If we choose time intervals to represent 1 s, that would be 180 time instants. If the entry point is w^{L} and the exit point is w^{R}, when we consider an entry time of t, we would create all the arcs

$$\big((w^{\mathrm{L}}, t), (w^{\mathrm{R}}, t+180)\big), \big((w^{\mathrm{L}}, t), (w^{\mathrm{R}}, t+181)\big), \big((w^{\mathrm{L}}, t), (w^{\mathrm{R}}, t+182)\big), \dots$$

up to the end of the time horizon. This level of accuracy is clearly not needed in this situation: a train that took 181 time instants rather than 180 to travel along that segment, would go at a speed of 99.45 km/h which is indistinguishable from 100km/h for any practical purpose. So, even if removing some edge from the graph would — in principle — cause the algorithm to miss some feasible train plan, if these edges are properly selected, we can easily reduce the probability to miss a train plan that would produce a considerable improvement.

Here we propose four main strategies for graph sparsification. Let v be a node in the network, i the train under consideration and $m_{i,v}$ the minimum travel time of i along v. Furthermore, let t be the entry time of the train at the node and t' the first feasible exit time, taking into account both the minimum travel time and the other constraints such as the minimum and maximum out-times $t^{\mathrm{out}}_{i,v}$ and $T^{\mathrm{out}}_{i,v}$. The strategies we used are the following:

Fixed step We only consider departure times starting at t' and then keeping a time instant in every s. The set of possible departure times will be

$$\left\{ t' + k \cdot s \;\middle|\; k = 0, 1, \dots, \right\}$$

Fixed step with threshold The previous strategy can be improved by specifying a threshold τ and keeping all departure times between t' and $t' + \tau$. In this way we keep those arcs that are close to t' and therefore correspond to a minimal delay of the train.

Linear This case is similar to the fixed step sparsification, but the step s is adjusted to be inversely proportional to $m_{i,v}$. The set of possible departure times is

$$\left\{ t' + k \cdot \max\left(1, \frac{m_{i,v}}{s}\right) \;\middle|\; k = 0, 1, \dots, \right\}$$

Progressive With this strategy we allow a higher density for time instants close to t', while we retain fewer arcs as long as we move away from that time instant. The idea behind this criterion is that good train plans are characterised by train schedules that are delayed as little as possible. The set of possible departure times is $\{t'_k\}$ where $t'_0 = t'$ and

$$t'_k = t'_{k-1} + 1 + \left\lfloor \frac{t'_{k-1} - t}{s} \right\rfloor$$

In our testing we used $s \in \{2, 3, 5\}$ and $\tau \in \{5, 10, 15\}$, leading to a total of 19 combinations, including the case when sparsification is disabled. Fig. 7a to d give a graphical representation of the sparsification methods for a segment v with endpoints w^{L}_1 and w^{R}_2 and a minimum travel time $m_{i,v} = 2$.

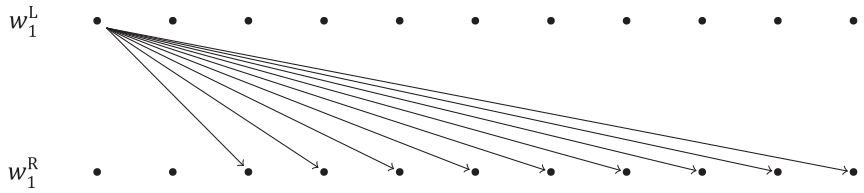
6. Computational results

The aim of computational results presented in this section is to verify that the proposed approach is valid for complex instances coming from real-life applications, that describe an extended network with many trains, and an extended time horizon with a dense discretisation. We also want to validate that our algorithm achieves good results in real-time settings, where running times are limited to few seconds.

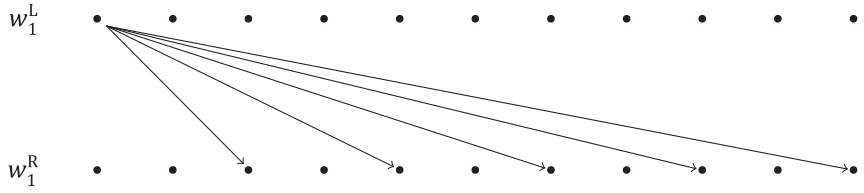
To this end, we have initially considered 23 instances, generated from three different real-world railway networks, provided to us by Alstom Transport. The first set of instances (N01-N13) describes a relatively small network characterised by the presence of single track lines used in both directions. The second set of six instances (L01-L06) refers to a busy regional network with a large main station and several smaller stations. Finally, the third set includes four instances (P01-P04) describing a high-speed network with frequent long-distance trains.

Table 1 outlines the main characteristics of the instances considered. Column "# Trains" lists the number of trains present in the instance; "# Nodes" is the number of resources, i.e., the cardinality of the set V in the network graph G_{N} (before time expansion); "Time horizon" is the span, in hours, of the planning horizon; "Discretisation" is the time discretisation step used, expressed in seconds; "# Conflicts" is the sum of the number of conflicts (as described in Section 4.3: illegal overtaking, hard capacity violation, headway time violation) plus the number of violated mandatory time dependencies, given as input in the current train plan.

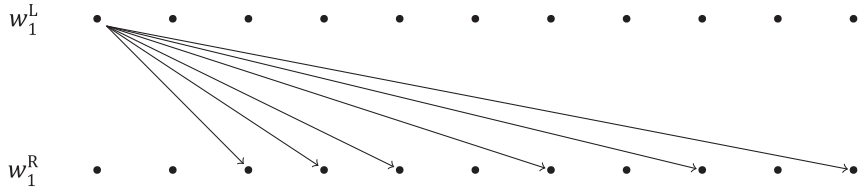
In Section 6.1 we perform parameter tuning, to determine which graph sparsification methods and initial sortings are more likely to produce good solutions when used together with each policy (RVNS or Tabu) and time limit (2s or 10s). In Section 6.2 we run a simple parallel version of the algorithm on the 23 instances, using the tuned parameters.



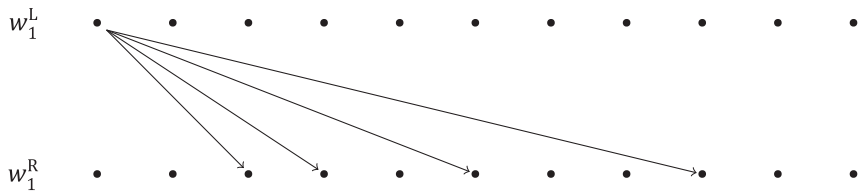
(a) No sparsification.



(b) Fixed step sparsification with $s = 2$.



(c) Fixed step sparsification with $s = 2$ and threshold $\tau = 3$.



(d) Progressive sparsification.

Fig. 7. Graphical representation of various graph sparsification techniques on the time-space graph.

In order to validate and benchmark our approach, we also generated new instances, which we are making publicly available. We used the network topology of the 2012 RAS Competition instances (see INFORMS Railways Applications Section, 2012), in two configurations: in the first (letter "N"), each segment is modelled as a separate resource, giving an N-track scenario; in the second (letter "S") parallel tracks are not modelled separately, but as a single resource with the appropriate capacity, giving a single-track scenario. We generated 15 instances of each type, divided in groups of 3. The nominal timetable is the same for each group, while the disturbances change, so to have 5 different forecast timetables for each nominal one. Because the RAS network is smaller than the networks used in the instances of Table 1, in order to obtain feasible nominal timetables we had to either use fewer trains (instances of the first group), or a longer time horizon with a more coarse discretisation (instances of the second and third groups). These instances are available at Santini (2017).

To ease the comparison with other algorithms that might be developed in the future, we did not consider problem-specific features such as time dependencies, and splits and merges. The conflicts that can arise, therefore, are limited to headway, hard and soft capacity, crossing, and overtake violations. Soft violations are penalised with a simple linear function. Table 2 describes the features of the generated instances; columns "H", "O", and "C" give a detailed breakdown of the type of conflicts: headways, overtake, and hard capacity, respectively. In Section 6.2 we apply the tuned parallel algorithm to the new instances, similarly to what we do for the proprietary instances.

Table 1
Main characteristics of the instances provided by Alstom.

Name	# Trains	# Nodes	Time horizon (h)	Discretisation (s)	# Conflicts
N01	28	108	2	15	24
N02	16	167	2	15	15
N03	28	172	2	15	36
N04	17	112	2	15	4
N05	18	112	2	15	12
N06	17	112	2	15	2
N07	28	126	2	15	12
N08	30	132	2	15	5
N09	28	130	2	15	4
N10	30	135	2	15	3
N11	15	137	1	15	1
N12	20	153	1	15	2
N13	33	135	4	15	37
L01	139	664	1	15	20
L02	103	631	0.75	15	54
L03	131	666	1	15	62
L04	132	675	1	15	25
L05	151	673	1.25	15	97
L06	133	671	1	15	38
P01	55	859	1	10	22
P02	55	814	1	10	22
P03	61	742	1	10	72
P04	71	731	1	10	70

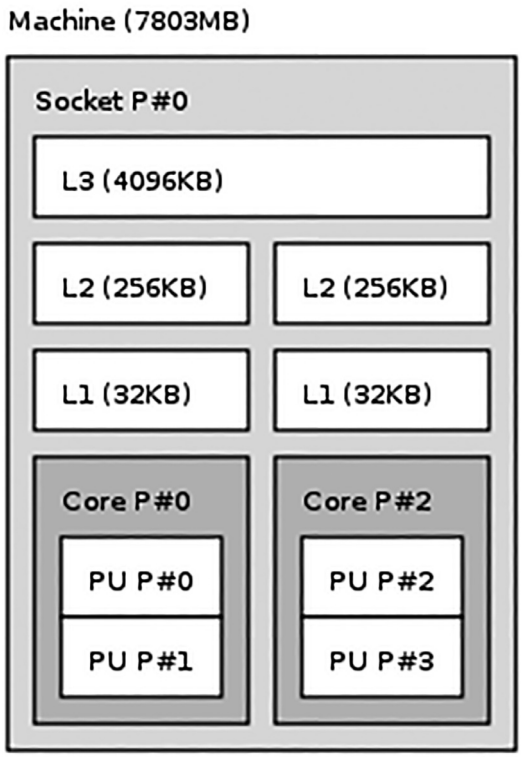


Fig. 8. Schematic hardware configuration of the test machine.

The tests presented in this section have been run on a dual-core 3.2GHz Intel i5 machine, with 7803 MB of RAM. The CPU configuration and the L1, L2 and L3 cache sizes are detailed in Fig. 8, as produced by the software Hwloc (Broquedis et al., 2010).

Table 2
Main characteristics of the instances generated starting from the 2012 RAS Competition instances.

Name	# Trains	Time horizon (h)	Discretisation (s)	# Conflicts		
				H	O	C
N-1-1	7	4	15	0	0	6
N-1-2	7	4	15	63	0	43
N-1-3	7	4	15	35	2	41
N-1-4	7	4	15	35	1	42
N-1-5	7	4	15	0	0	32
N-2-1	12	12	60	2	0	3
N-2-2	12	12	60	4	0	6
N-2-3	12	12	60	2	0	4
N-2-4	12	12	60	0	0	7
N-2-5	12	12	60	6	0	13
N-3-1	24	12	60	4	0	25
N-3-2	24	12	60	105	0	53
N-3-3	24	12	60	0	0	6
N-3-4	24	12	60	103	0	89
N-3-5	24	12	60	0	0	10
S-1-1	7	4	15	0	0	3
S-1-2	7	4	15	32	0	22
S-1-3	7	4	15	4	1	28
S-1-4	7	4	15	30	1	24
S-1-5	7	4	15	2	0	19
S-2-1	12	12	60	0	0	2
S-2-2	12	12	60	2	0	2
S-2-3	12	12	60	6	0	5
S-2-4	12	12	60	0	0	4
S-2-5	12	12	60	2	0	4
S-3-1	24	12	60	0	0	21
S-3-2	24	12	60	59	0	25
S-3-3	24	12	60	0	0	4
S-3-4	24	12	60	60	0	60
S-3-5	24	12	60	2	0	12

6.1. Parameter tuning

The objective of parameter tuning is to measure the impact of the sparsification methods and the sorting criteria introduced in Section 5, on real-time applications of the algorithm.

We ran six sets of experiments overall, in order to determine which combinations of sparsification and sorting are particularly effective with the RVNS and Tabu policies. For each of these two policies, the three sets of experiments only vary in the hard time limit given to the algorithm. The first two time limits are of 2 and 10 s (real-time); the third time limit is of 60 s, and is used to provide better solutions that can be used as a baseline for comparisons.

For each set of experiments, the tests were run on all the 23 Alstom instances. For each instance, we tried all combinations of sparsification methods and initial sortings, using the parameters described in Section 5. In total, we had 19 possible settings for the sparsification methods and 7 possible sortings, giving 133 tests for each policy, time limit and instance, giving a grand total of $133 \cdot 2 \cdot 3 \cdot 23 = 18\,354$ runs.

Tables 3 and 4 show the results we obtained during parameter tuning for the two solvers, with time limits 2 and 10 s. In the first table the results have been aggregated by sparsification method, while in the second, they have been aggregated by sorting criterion.

Columns “Sparsification” (in Table 3) or “Sorting” (in Table 4) tell, respectively, for which sparsification method or sorting criterion the data is being aggregated. For each line the data are grouped in four blocks, corresponding to the four combinations of policy (Tabu or RVNS) and time limit (2s or 10 s). Column “CF” gives the fraction of tests for which the algorithm was able to find a conflict-free schedule. Column “Dev” is the average deviation, calculated as $(z - z^*)/z^*$ where z is the solution value obtained by the algorithm with the specified configuration, and z^* is the best known solution value. This best known value comes from the 60 s runs that we use as baseline, and is the best value across all possible parameter combinations. Since, as explained in Section 5, the objective function has a hierarchical structure and unresolved conflicts take a very large penalty, the values in this column tend to be quite large, as one single instance for which a method was not able to produce a conflict-free schedule can increase the average considerably. This is, however, a good metric of the desirability of a method, because solving conflicts always has priority on any other measure of solution quality. In column “CF Dev”, we similarly report the average deviation, but this time we only consider the conflict-free solutions in the average.

Table 3
Parameter tuning results aggregated by sparsification method.

	Tabu 2s			Tabu 10s			RVNS 2s			RVNS 10 s		
Sparsification	CF	Dev	CF Dev	CF	Dev	CF Dev	CF	Dev	CF Dev	CF	Dev	CF Dev
disabled	0.64	10912.64	1.88	0.78	593.90	1.33	0.79	488.43	10.32	0.80	370.22	1.43
fixed-2	0.70	16252.12	1.68	0.77	15374.42	1.35	0.75	15811.52	1.73	0.75	15558.96	1.38
fixed-3	0.71	16161.95	1.66	0.78	15312.86	1.34	0.75	15686.94	1.62	0.76	15528.27	1.42
fixed-5	0.74	15875.32	1.62	0.78	15321.86	1.32	0.74	28820.40	1.59	0.75	26406.56	1.40
linear-2	0.65	6391.54	1.85	0.80	339.41	1.36	0.80	377.08	10.06	0.80	370.23	1.45
linear-3	0.64	8725.04	2.06	0.78	418.67	1.39	0.79	599.52	10.20	0.80	370.24	1.46
linear-5	0.62	11325.47	1.95	0.77	739.40	1.43	0.78	4981.04	8.29	0.80	370.25	1.47
progressive-2	0.76	834.95	1.49	0.82	31.93	1.24	0.78	696.63	1.51	0.80	308.66	1.34
progressive-3	0.73	1348.76	1.41	0.81	62.66	1.23	0.80	493.27	1.55	0.80	247.19	1.34
progressive-5	0.73	1439.36	1.50	0.82	64.21	1.23	0.78	663.47	1.57	0.78	379.93	1.32
threshold-2-5	0.66	3977.49	4.29	0.80	217.94	1.26	0.80	406.13	7.95	0.80	370.16	1.38
threshold-2-10	0.67	3851.51	4.25	0.79	247.15	1.27	0.78	445.13	8.05	0.79	339.43	1.38
threshold-2-15	0.66	4157.31	4.34	0.79	310.16	1.27	0.80	406.14	7.95	0.80	370.17	1.38
threshold-3-5	0.70	1482.97	1.62	0.80	247.14	1.28	0.79	568.01	7.93	0.80	370.15	1.36
threshold-3-10	0.69	1627.55	1.63	0.80	247.15	1.29	0.78	475.82	7.99	0.79	400.89	1.36
threshold-3-15	0.68	1742.63	4.20	0.80	187.18	1.24	0.79	436.84	7.95	0.79	370.16	1.36
threshold-5-5	0.73	1333.08	1.49	0.81	93.41	1.22	0.79	556.38	3.63	0.79	308.66	1.34
threshold-5-10	0.70	1708.34	1.56	0.81	93.42	1.24	0.79	525.67	3.67	0.79	370.14	1.34
threshold-5-15	0.69	1750.12	1.58	0.81	62.70	1.27	0.79	525.69	3.69	0.79	400.89	1.36

Table 4
Parameter tuning results aggregated by initial sorting.

	Tabu 2s			Tabu 10s			RVNS 2s			RVNS 10s		
Sorting	CF	Dev	CF Dev	CF	Dev	CF Dev	CF	Dev	CF Dev	CF	Dev	CF Dev
Congestion	0.73	3197.61	1.58	0.80	2509.98	1.29	0.79	3535.41	11.38	0.80	3349.47	1.42
Conflict time	0.73	7938.66	4.92	0.80	2589.95	1.28	0.78	4012.83	9.14	0.78	3700.91	1.31
Length	0.73	3302.75	1.64	0.80	2635.49	1.27	0.78	3882.08	1.75	0.78	3791.12	1.44
Random	0.59	13031.87	2.01	0.81	2637.88	1.34	0.76	5482.95	9.27	0.78	2623.51	1.45
Reverse congestion	0.74	5170.65	1.72	0.78	2589.07	1.29	0.78	3848.59	1.77	0.78	3756.94	1.37
Reverse speed	0.65	4110.60	1.61	0.79	2584.59	1.26	0.77	2890.27	1.75	0.77	2841.63	1.35
Speed	0.68	4105.06	1.70	0.79	2861.43	1.31	0.82	3229.39	4.74	0.82	3224.74	1.33

We want to select the best sparsification method for each policy (RVNS or Tabu) and each time limit (2 s or 10 s). In order to do this, it is not sufficient to take the policy with the lowest deviation, but one has to ensure that the differences in deviation are statistically relevant.

For this reason, we ran a Wilcoxon signed-rank test on each pair of sparsification methods, to measure whether their deviations across the various instances come from the same distribution or not (in this latter case, a difference in the average deviation is statistically relevant).

Figs. 9 and 10 give a graphical representation of the outcomes of the Wilcoxon test. Each figure gives a representation of a directed graph. Sparsification methods are represented as nodes. An arc is drawn between two nodes if the p-value of the Wilcoxon test relative to two sparsifications is < 0.05; the arc goes from the node with the better average deviation to that with the worse one. The colour and the thickness of the arc depend on the difference between the deviations: the greater the difference, the thicker and bluer the arc; on the other hand, small differences are represented by thin red-ish arcs. In particular, notice that the thickness and colour of the arc are not related with the p-value, which is used only as a threshold to decide whether to create the arc.

To simplify the visualisation, nodes and arcs are drawn from top to bottom, so that the best sparsifications are in the top part of the graph. Furthermore, in order to reduce the number of arcs drawn in the figure, whenever there are arcs (M_1, M_2), (M_2, M_3) and (M_1, M_3), this latter arc is removed. In other terms, if a method M_1 dominates method M_2, and M_2 dominates M_3, it is implicit that M_1 dominates M_3.

The sparisification methods were chosen as follows:

- In the case of policy Tabu and time limit 2 s (Fig. 9a) the only two undominated methods are “progressive-2” and “progressive-3”; since the average deviation of the former is 38.1% smaller than that of the latter (see Table 3), we decided to take “progressive-2” as the chosen sparsification method.
- For policy RVNS and time limit 2s, the only undominated method is “linear-2”, which also has a considerably smaller deviation compared to the other methods.
- An interesting case is that of policy Tabu and time limit 10 s (Fig. 10a), as this is the case where it is most unclear which sparsification emerges as a winner. However, since the deviation of method “progressive-2” is the smallest, and it is 49.04% smaller than the second-smallest one (“progressive-3”), we chose this method.

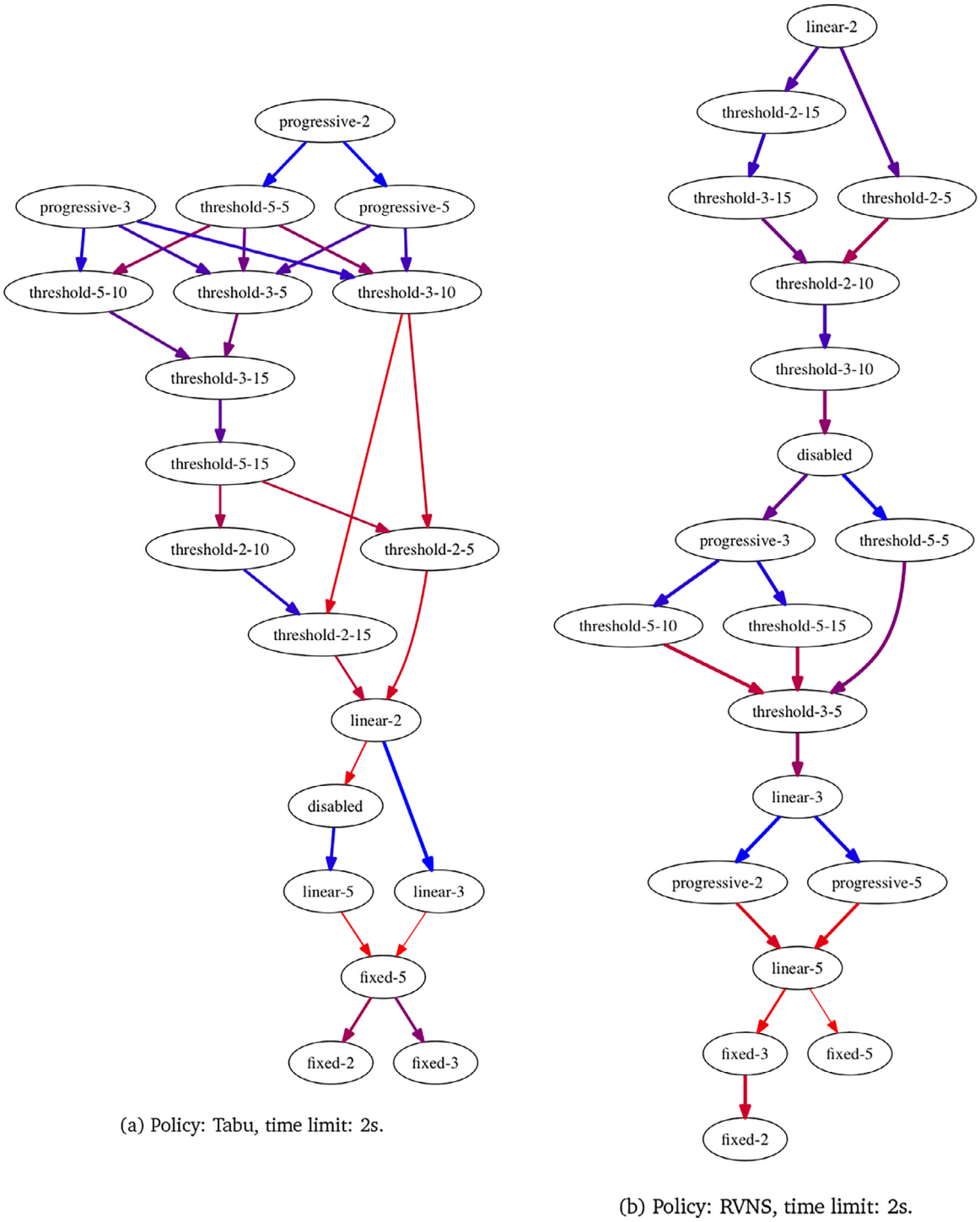
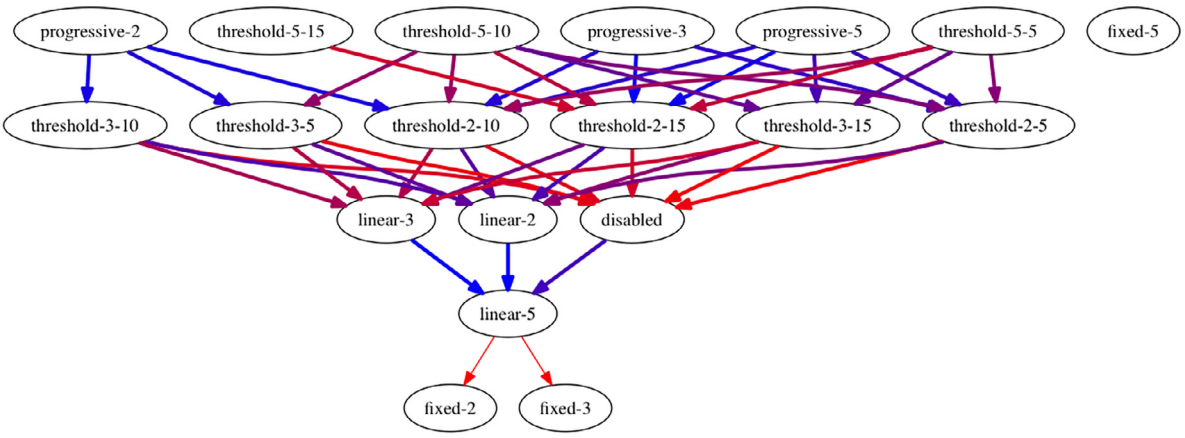


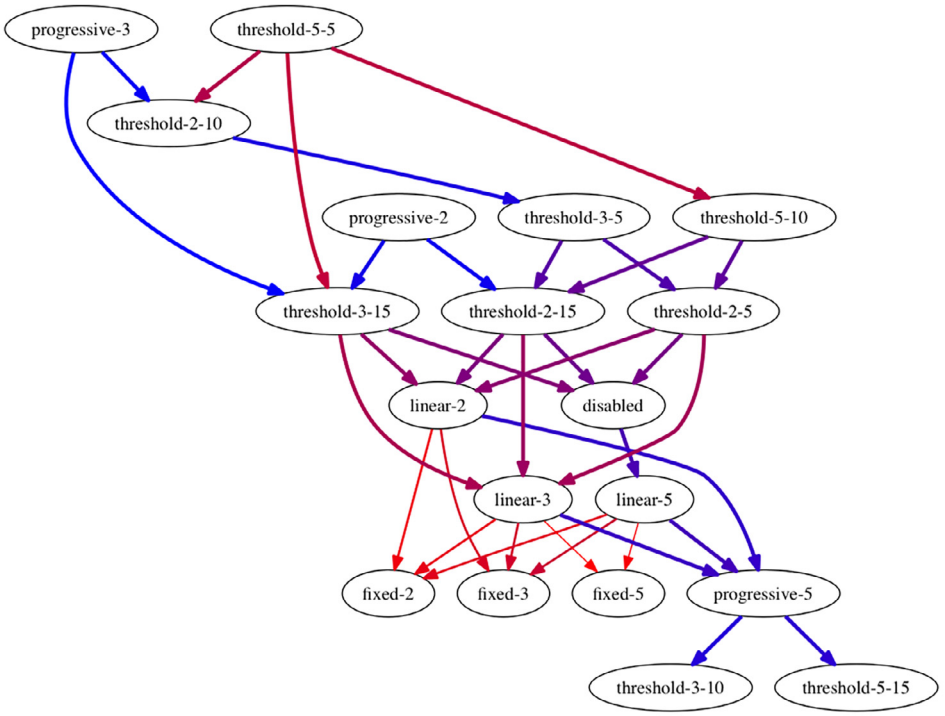
Fig. 9. Visual representation of the results of the Wilcoxon test (time limit: 2 s).

- Finally, for policy RVNS and time limit 10s, the three undominated methods were “progressive-2”, “progressive-3”, and “threshold-5-5”. Again, since the average deviation for “progressive-3” is 19.92% smaller than that for the other two (which have the same deviation), we chose that method.

Table 5 shows the chosen sparsification methods and gives the average deviations (columns “Avg”), together with their standard deviation (column “Std”), obtained by employing the different initial sortings with each sparsification. Column



(a) Policy: Tabu, time limit: 10s.



(b) Policy: RVNS, time limit: 10s.

Fig. 10. Visual representation of the results of the Wilcoxon test (time limit: 10s).

"Best" tells the number of instances (out of 23) for which each sorting criterion provided the best result, compared to the other criteria in the same column.

6.2. Parallel algorithm

In this section we provide computational results for a very simple parallel implementation of the algorithm. We ran four sets of experiments, namely one for each combination of policy and time limit, together with the respective sparsification method chosen during parameter tuning, as described in Section 6.1. For each set, the parallel implementation simply consists of launching four parallel instances of the algorithm, each using one of four sorting criteria. When the time limit hits, the four solutions provided by the parallel instances are examined and the best one is returned as the overall solution.

Table 5
Average deviations of different sortings, for the chosen sparsification methods.

	Tabu 2s progressive-2 Deviation			Tabu 10s progressive-2 Deviation			RVNS 2s linear-2 Deviation			RVNS 10s progressive-3 Deviation		
Sorting	Avg	Std	Best	Avg	Std	Best	Avg	Std	Best	Avg	Std	Best
Congestion	441.45	2115.65	8	0.16	0.22	12	11.83	53.76	6	0.36	0.83	8
Conflict time	430.66	2063.77	2	0.22	0.30	2	442.05	2062.05	6	430.57	2063.81	5
Length	1076.20	5159.38	4	0.24	0.38	2	861.38	4127.40	2	215.53	1031.94	1
Random	2461.98	9855.31	3	0.22	0.31	2	227.90	1030.85	1	430.60	2063.80	3
Rev. Congestion	226.44	1083.85	3	0.14	0.16	2	646.11	3095.51	3	430.59	2063.73	1
Rev. Speed	554.88	1887.01	1	0.19	0.33	2	431.11	2063.69	3	215.47	1032.00	1
Speed	646.03	3095.86	2	215.37	1031.95	3	12.15	53.70	2	0.23	0.42	4

Table 6
List of sorting criteria chosen for each policy and time limit, to be used in the parallel algorithm.

Tabu 2s progressive-2	Tabu 10s progressive-2	RVNS 2s linear-2	RVNS 10s progressive-3
Congestion	Congestion	Congestion	Congestion
Length	Conflict time	Conflict time	Conflict time
Reverse congestion	Reverse Congestion	Reverse congestion	Speed
Random	Reverse speed	Reverse speed	Random

The usage of parallel algorithms in operational research is well-established. We refer the reader to, e.g., Clausen and Perregaard (1999) for parallel strategies for branch-and-bound exact algorithms, or to Ropke and Santini (2016) for a systematic analysis of the speed-ups obtained by parallelising the Adaptive Large Neighbourhood Search metaheuristic. With respect to the train rescheduling problem, Iqbal et al. (2012); 2013) proposed parallel algorithms for rescheduling under disturbances.

In our case, the implementation of a parallel algorithm is motivated by the high dispersion of the solution values with respect to the average one, obtained by the different sorting methods, as witnessed by the high values of Standard Deviation reported in Table 5 (this effect is more evident for the 2s time limit compared to the 10s one, and for the RVNS policy compared to the Tabu one). This means that, in practice, even the best sorting method was not able to resolve some solvable conflict, thus resulting in high solution values, due to the hierarchical nature of our objective function. Furthermore, we observed a certain complementarity in the capability of resolving conflicts across different sortings, which we see as a hint towards the parallel use of different sorting criteria.

More formally, we investigated the dependance of sorting criteria to instance characteristics via simple *one-vs-all* and *one-vs-one* multiclass classification algorithms provided by the library `scikit` (see, e.g., Aly, 2015). These algorithms were based on the binary classifier that, for a fixed time limit, assigns an instance to an initial sorting criterion (class) if there is at least one policy (either Tabu or RVNS) for which that sorting provided the best result for that time limit. The instance features considered were the one listed in Tables 1 and 2. Neither the *one-vs-all* nor the *one-vs-one* algorithm found statistically significant relationships of the sorting criteria to the instance features.

Once established the need for an algorithm that employs more than one sorting criteria at once, it is clearly important to perform a good choice of the criteria. The simplest approach would be to choose the four criteria that produce the four lowest deviations for a given policy and time limit (see Table 5). This choice, however, has not proven particularly good especially for the lowest time limit, and in one case we even had one instance (instance "P4" for the Tabu policy at 2 s) for which not all solvable conflicts were actually resolved.

What we aim for is a choice of methods out of which, given any instance, there are high chances to find one that works reasonably well with that instance, eliminating all solvable conflicts, and therefore exploiting the aforementioned complementarity. For this reason, we decided to choose the methods in a way to maximise the number of instances for which at least one of the methods chosen was the best. Table 6 lists the chosen sorting criterion for each policy and time limit.

Table 7 reports aggregate results for the parallel algorithm on the Alstom instances. Column "Sorting" reports which initial sorting produced the optimal solution most often, for a fixed choice of instance group and algorithm. Column "Dev" gives the average deviation, calculated as $(z - z^*)/z^*$ where z is the solution value obtained by the algorithm, and z^* is the best known solution value. Column "Conflicts" lists the average number of conflicts remaining in the output solution, while columns "Infeasible" and "Modified" report, respectively, the number of trains that are infeasible and whose schedule has been modified in the output solution. Finally, "Delay" lists the average delay (or advance, in which case the figure is < 0) reported by trains at their final destination.

By observing the tables, it is clear that the benefit of the parallel algorithm is considerable when compared to fixed choices of sorting criteria. This is particularly evident for the 2 s time limit, where the best average deviations achieved

Table 7
Parallel algorithm results on the Alstom instances.

Instance group	Algorithm	Sorting	Dev	Conflicts	Infeasible	Modified	Delay
L	Tabu 2s	Random, Length, Reverse congestion	0.17	0.50	0.00	104.50	28.64
L	Tabu 10s	Reverse congestion	0.11	0.50	0.00	104.67	29.00
L	RVNS 2s	Reverse speed	0.17	0.50	0.00	105.67	33.34
L	RVNS 10s	Conflict time	0.05	0.50	0.00	100.83	10.64
N	Tabu 2s	Congestion	0.16	0.38	0.08	7.85	79.43
N	Tabu 10s	Congestion	0.08	0.38	0.08	8.23	70.12
N	RVNS 2s	Congestion	0.45	0.38	0.08	7.54	99.30
N	RVNS 10s	Congestion	0.26	0.38	0.08	8.00	67.69
P	Tabu 2s	Reverse congestion	0.50	0.00	0.00	59.25	247.17
P	Tabu 10s	Reverse speed	0.03	0.00	0.00	59.50	220.52
P	RVNS 2s	Conflict time	0.30	0.00	0.00	59.50	226.83
P	RVNS 10s	Congestion	0.20	0.00	0.00	59.50	221.58
Overall 2s		**Congestion**	**0.29**	**0.35**	**0.04**	**42.09**	**99.81**
Overall 10s		**Congestion**	**0.14**	**0.35**	**0.04**	**41.74**	**82.56**
Overall Tabu		**Congestion**	**0.15**	**0.35**	**0.04**	**42.15**	**90.45**
Overall RVNS		**Congestion**	**0.27**	**0.35**	**0.04**	**41.67**	**91.92**
Overall		**Congestion**	**0.21**	**0.35**	**0.04**	**41.91**	**91.19**

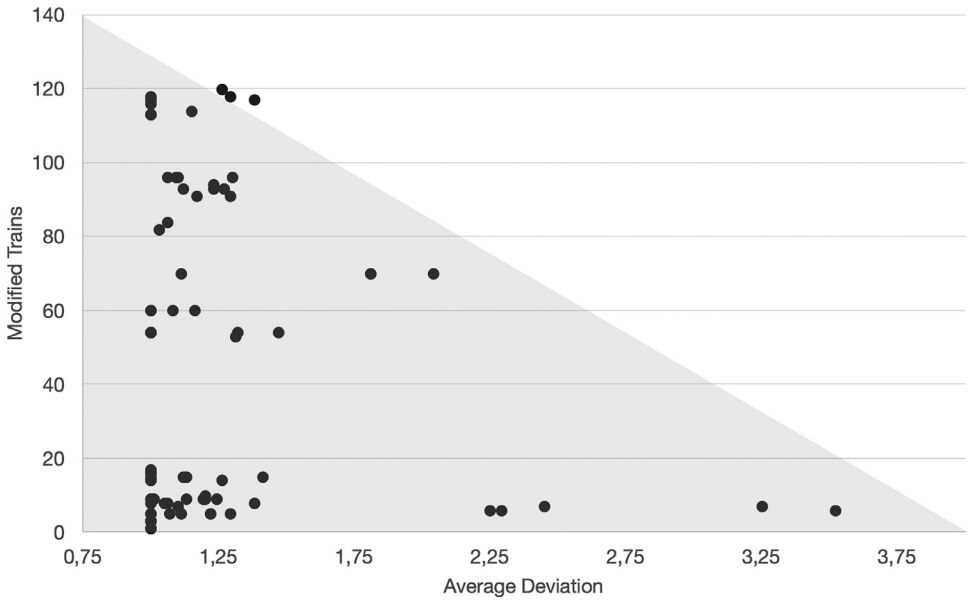


Fig. 11. Scatter graph (without outliers) showing the correlation between solution quality and number of modified trains.

by using only one sorting (see Table 5) were of 226.44 for Tabu and 11.83 for RVNS, while the parallel algorithms gives — respectively — 0.22 and 0.35 (see the detailed results in Tables 9 and 10). In addition, by looking at the detailed results, we can notice that all four sortings selected for each set (algorithm and time limit) provide the best solution in some instances, with the "Congestion" sorting being the one appearing most frequently overall, and also when aggregating by policy or by time limit.

For what concerns the choice of the heuristic policy, from the detailed results we can notice that Tabu and RVNS turn out to provide the smallest instance-by-instance deviations almost the same number of times: namely, both 15 times for 2s, and 18 vs 17 times for 10 s. From Table 7 we can see, however, that Tabu provides smaller deviations, at the cost of modifying more trains. A small further reduction could be achieved by an hypothetical algorithm that ran the Tabu and RVNS policies in parallel (thereby using 8 concurrent threads): such an algorithm would achieve an average deviation of 0.17 for 2s, and 0.06 for 10s. Finally, as we clearly expected, a considerable improvement is obtained by letting the algorithm run for longer: the average deviation for all methods ran for 10s is less than half than that for all methods ran for 2s.

Fig. 11 displays the relation between the deviations achieved by the parallel algorithm and the number of trains whose schedules have been modified. All the solutions described in Tables 9 and 10 are reported in the figure. The figure seems to suggest that, despite not having included the number of modified trains as a penalty term in the objective function (see Section 4.4.5) there are no solutions in which a lot of trains are modified and, despite that, bad solutions are obtained. This can be seen by noticing that the upper-right triangle of the graph is empty. In summary, the figure seem to suggest

Table 8
Parallel algorithm results on the RAS-Based instances.

Instance group	Algorithm	Sorting	Dev	Conflicts	Infeasible	Modified	Delay
S-1	Tabu 2 s	Reverse congestion	$2.26 \cdot 10^{-2}$	0	0	4.60	132.67
S-1	Tabu 10 s	Reverse congestion	$1.99 \cdot 10^{-2}$	0	0	4.60	131.72
S-1	RVNS 2 s	Reverse speed	$3.58 \cdot 10^{-2}$	0	0	4.60	134.77
S-1	RVNS 10 s	Length, Speed	$1.64 \cdot 10^{-2}$	0	0	4.60	129.30
S-2	Tabu 2 s	Reverse congestion	$3.97 \cdot 10^{-3}$	0	0	3.80	63.77
S-2	Tabu 10 s	Congestion	$1.83 \cdot 10^{-3}$	0	0	3.80	60.61
S-2	RVNS 2 s	Conflict time	$6.75 \cdot 10^{-4}$	0	0	3.80	60.04
S-2	RVNS 10s	Speed	$3.54 \cdot 10^{-4}$	0	0	3.80	59.80
S-3	Tabu 2 s	Length, Congestion	$1.24 \cdot 10^{-2}$	0	0	5.20	185.33
S-3	Tabu 10 s	Congestion	$2.65 \cdot 10^{-3}$	0	0	5.00	188.68
S-3	RVNS 2 s	Conflict time	$7.52 \cdot 10^{-3}$	0	0	5.20	190.85
S-3	RVNS 10 s	Congestion, Speed	$8.02 \cdot 10^{-5}$	0	0	5.20	186.63
N-1	Tabu 2 s	Random, Congestion	$6.91 \cdot 10^{-4}$	0	0	4.20	125.05
N-1	Tabu 10 s	Reverse congestion	$6.91 \cdot 10^{-4}$	0	0	4.20	125.05
N-1	RVNS 2 s	Conflict time	$1.19 \cdot 10^{-1}$	0	0	3.80	154.79
N-1	RVNS 10 s	Congestion	$2.25 \cdot 10^{-4}$	0	0	4.20	124.86
N-2	Tabu 2 s	Length	$9.90 \cdot 10^{-4}$	0	0	4.00	106.31
N-2	Tabu 10 s	Conflict time	$9.90 \cdot 10^{-4}$	0	0	4.00	106.31
N-2	RVNS 2 s	Conflict time	$7.52 \cdot 10^{-4}$	0	0	4.00	106.42
N-2	RVNS 10 s	Length	$1.32 \cdot 10^{-4}$	0	0	4.00	106.06
N-3	Tabu 2 s	Reverse congestion	$2.15 \cdot 10^{-2}$	0	0	6.00	219.10
N-3	Tabu 10 s	Conflict time	$7.14 \cdot 10^{-3}$	0	0	5.40	203.28
N-3	RVNS 2 s	Conflict time	$9.28 \cdot 10^{-3}$	0	0	5.20	205.11
N-3	RVNS 10 s	Speed	$7.40 \cdot 10^{-3}$	0	0	5.20	207.49
Overall 2s		**Conflict time**	$\mathbf{1.93 \cdot 10^{-2}}$	**0**	**0**	**4.20**	**140.27**
Overall 10s		**Congestion**	$\mathbf{5.07 \cdot 10^{-3}}$	**0**	**0**	**4.23**	**135.90**
Overall Tabu		**Reverse congestion**	$\mathbf{7.05 \cdot 10^{-3}}$	**0**	**0**	**4.30**	**137.32**
Overall RVNS		**Conflict time**	$\mathbf{1.64 \cdot 10^{-2}}$	**0**	**0**	**4.20**	**138.84**
Overall		**Conflict time**	$\mathbf{1.22 \cdot 10^{-2}}$	**0**	**0**	**4.25**	**138.08**

that three scenarios can happen. The first, best scenario is that a high quality solution is found and few trains are modified (bottom-left cluster of points); in the second scenario a high quality solution is found, but a lot of trains have to be modified (points on the top-left corner); finally, rarely a bad solution is found, but in this case only few trains are modified (bottom-right points).

Table 8 is analogous to Table 7 and reports aggregate results for the parallel algorithm on the RAS-based instances. Notice that, due to the nature of the instances, the deviation are smaller compared to those reported in Table 7, and all conflicts were resolved in each instance. Also, the average number of modified trains is smaller for RAS instances than for Alstom instances; this is not surprising, as the number of train in the nominal timetable was also smaller. Average delays are, on the other hand, higher, probably due to the fact that the time horizon is longer for the RAS-based instances.

As in the case of the Alstom instances, there is a lot of variability in which initial sorting criterion leads to the best solution; not only some criterion works best with particular instances, but also the combination of instance and policy seems to influence the effectiveness of the sorting criterion. This confirms the negative results obtained by the classification algorithms, and the potential of a (simple) parallel implementation in order to obtain good practical results.

7. Conclusions

In this paper we presented a fast algorithm for the conflicts resolution in real-time train traffic management. The development of the algorithm was motivated by an extensive collaboration between the authors and Alstom, and an industrial version of it is currently integrated into the Train Management System ICONIS and will be deployed at various Alstom's customers worldwide.

The algorithm is designed to combine speed and flexibility through an abstract modelling of the rail infrastructure that can be used both at a microscopic and at a macroscopic (i.e. aggregated) representation of the network, and even a mix of the two. Overall the algorithm is based on the repeated execution of a greedy approach which schedules trains on a time-spaced network and several different dispatching rules. Two different shaking techniques, one based on Reduced Variable Neighborhood Search and the other on Tabu Search, are implemented. The speed required to handle realistic sized instances, involving tens of trains during a time horizons of a few hours, is achieved by implementing different effective sparsification techniques of the time-space graph which greatly reduce the computational effort while preserving good accuracy of the results. In addition, an extensive study of the impact of each algorithm component on the quality and speed of the algorithm was conducted, so as to identify some best performing parameter combinations which can be used inside a simple parallelisation scheme.

The algorithm is capable of handling the constraints typically encountered in real-world train rescheduling applications and, thanks to the time-space representation and the greedy nature of its main structure, can be easily adapted to accommodate possible further characteristics. Furthermore, a very general hierarchic objective function allows to optimize different aspects of the problem as the number of possibly remaining conflicts, the average delay and the number of modifications with respect to the nominal timetable.

An extensive computational testing has been performed by using both real-world instances provided by Alstom and artificial instances constructed from existing benchmarks from the literature. The main aim of the testing was to identify the best performing components and parameter configurations and to validate the effectiveness of the algorithm for the real-time conflict resolution. The obtained results show that the proposed algorithms is, within a few seconds, consistently capable of resolving existing conflicts and obtain highly optimized solutions. This is also confirmed by the extensive testing of the industrial version of the algorithm performed by Alstom during the last years where the solutions were consistently found correct and effective by expert users when compared with traditionally employed priority dispatching rules.

Even though some of the components of our algorithm were already known in the literature, we think that the main contribution of our approach is the effective combination of various dispatching rules with shaking, together with the use of sparsification and parallelisation to achieve high speed without compromising quality. Clearly the simple structure and the required speed may lead sometimes to difficulty in removing all existing conflicts (or detecting that they are not solvable) or in relatively poor solutions but the computational results show that this is a rare event and that the algorithm provides an excellent compromise between quality and speed.

Acknowledgements

The authors are grateful to Alstom Transport for providing the test instances used for the computational validation of our method.

Appendix A. Detailed results

Tables 9 and 10 provide instance-by-instance results for Tabu and RVNS policies, respectively. Column “Sorting” specifies which was the sorting method employed in the thread that produced the overall best solution. Column “Dev”, analogously to what presented in previous tables, is the ratio between the objective value produced by the parallel algorithm and the best known objective value for the same instance. Column “Conflicts” lists the number of hard and soft constraints violated in the produced solution. Column “Inf Trains” tells how many trains remain infeasible (i.e., with hard constraints violations). Column “Mod Trains” gives the number of trains whose schedules have been modified. Finally, column “Avg Delay” lists, in time units, the average delay (if >0) or advance (if <0) that a train reported when arriving at its destination. A † next to a deviation indicates the best deviation produced for the corresponding instance, for that time limit. When Tabu and RVNS attained the same deviation, the † is present in both tables.

Notice that there are instances for which no method was able to resolve all Conflict timelists (even when using a time limit of 60s). Manual inspection of these Conflict timelists has Conflict timerimed that they are, indeed, unavoidable.

Tables 11 and 12 are analogous to Tables 9 and 10, but provide detailed results relative to the RAS-based instances.

Table 9
Parallel algorithm results for the Tabu solver on the Alstom instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
Tabu 2s progressive-2	N1	Length	0.20	0	0	9	94.14
	N2	Length	†0.13	0	0	15	198.53
	N3	Reverse congestion	†0.00	3	1	8	40.71
	N4	Congestion	†0.00	0	0	5	15.00
	N5	Random	†0.00	0	0	16	102.63
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Length	†0.05	0	0	8	70.34
	N8	Congestion	†0.24	0	0	9	3.87
	N9	Congestion	†1.25	0	0	6	32.07
	N10	Congestion	0.22	0	0	5	9.19
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Congestion	†0.00	2	0	14	408.53
	L1	Random	†0.00	1	0	117	-13.71
	L2	Reverse congestion	†0.10	0	0	96	-39.17
	L3	Random	†0.00	2	0	113	87.02
	L4	Reverse congestion	0.29	0	0	91	33.66
	L5	Length	0.38	0	0	117	59.70

(*continued on next page*)

Table 9 (*continued*)

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
	L6	Length	0.27	0	0	93	44.33
	P1	Reverse congestion	0.47	0	0	54	243.21
	P2	Reverse congestion	†0.31	0	0	53	256.07
	P3	Congestion	0.16	0	0	60	141.05
	P4	Random	1.04	0	0	70	348.33
	Overall		0.22	0.35	0.04	42.00	95.35
Tabu 10s progressive-2	N1	Reverse speed	0.20	0	0	10	95.17
	N2	Conflict time	†0.00	0	0	15	192.35
	N3	Congestion	†0.00	3	1	9	49.29
	N4	Congestion	†0.00	0	0	5	15.00
	N5	Congestion	†0.00	0	0	16	102.63
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Congestion	†0.01	0	0	9	70.34
	N8	Congestion	†0.24	0	0	9	3.87
	N9	Congestion	†0.38	0	0	8	10.34
	N10	Congestion	†0.22	0	0	5	9.19
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Congestion	†0.00	2	0	14	305.74
	L1	Reverse congestion	†0.00	1	0	117	-23.14
	L2	Reverse congestion	0.09	0	0	96	-42.23
	L3	Conflict time	†0.00	2	0	113	106.49
	L4	Reverse congestion	0.17	0	0	91	28.85
	L5	Reverse speed	0.29	0	0	118	73.71
	L6	Congestion	0.12	0	0	93	30.34
	P1	Reverse speed	†0.00	0	0	54	233.84
	P2	Conflict time	†0.00	0	0	54	236.25
	P3	Congestion	†0.00	0	0	60	138.87
	P4	Reverse speed	†0.11	0	0	70	273.12
	Overall		0.08	0.35	0.04	42.30	85.55

Table 10
Parallel algorithm results for the RVNS solver on the Alstom instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
RVNS 2s linear-2	N1	Reverse congestion	†0.19	0	0	9	94.14
	N2	Reverse congestion	0.26	0	0	14	229.41
	N3	Reverse congestion	†0.00	3	1	9	48.75
	N4	Reverse congestion	0.07	0	0	5	15.00
	N5	Congestion	0.41	0	0	15	273.95
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Reverse speed	0.10	0	0	7	88.45
	N8	Conflict time	0.52	0	0	6	49.35
	N9	Conflict time	0.25	0	0	7	20.69
	N10	Conflict time	†0.11	0	0	5	6.77
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Congestion	†0.00	2	0	14	406.76
	L1	Conflict time	†0.00	1	0	118	-5.25
	L2	Conflict time	0.30	0	0	96	-17.48
	L3	Reverse speed	†0.00	2	0	113	81.07
	L4	Reverse congestion	†0.23	0	0	93	36.87
	L5	Reverse speed	†0.26	0	0	120	49.17
	L6	Reverse speed	†0.23	0	0	94	55.63
	P1	Congestion	†0.00	0	0	54	235.71
	P2	Conflict time	0.32	0	0	54	236.25
	P3	Conflict time	†0.08	0	0	60	150.97
	P4	Congestion	†0.81	0	0	70	284.38
	Overall		0.35	0.35	0.04	42.17	104.27

(*continued on next page*)

Table 10 (*continued*)

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
RVNS 10s progressive-3	N1	Random	†0.13	0	0	9	97.76
	N2	Random	0.12	0	0	15	198.53
	N3	Speed	†0.00	3	1	9	47.68
	N4	Congestion	†0.00	0	0	5	15.00
	N5	Conflict time	†0.00	0	0	16	101.05
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Conflict time	0.06	0	0	8	88.45
	N8	Speed	1.45	0	0	7	28.55
	N9	Speed	1.29	0	0	6	33.62
	N10	Congestion	0.29	0	0	5	9.68
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Random	†0.00	2	0	17	202.06
	L1	Conflict time	†0.00	1	0	116	-21.96
	L2	Congestion	†0.06	0	0	96	-46.75
	L3	Conflict time	†0.00	2	0	113	60.34
	L4	Conflict time	†0.03	0	0	82	19.01
	L5	Congestion	†0.15	0	0	114	14.21
	L6	Speed	†0.06	0	0	84	38.96
	P1	Congestion	†0.00	0	0	54	232.50
	P2	Conflict time	†0.00	0	0	54	234.91
	P3	Random	†0.00	0	0	60	134.52
	P4	Congestion	0.81	0	0	70	284.38
	Overall		0.19	0.35	0.04	41.17	79.57

Table 11
Parallel algorithm results for the Tabu solver on the RAS-based instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
Tabu 2s progressive-2	S-1-1	Length	$4.10 \cdot 10^{-2}$	0	0	5	200.20
	S-1-2	Reverse congestion	†0	0	0	4	109.23
	S-1-3	Length	$3.32 \cdot 10^{-2}$	0	0	5	78.11
	S-1-4	Reverse congestion	$2.55 \cdot 10^{-2}$	0	0	5	186.38
	S-1-5	Reverse congestion	†0	0	0	4	84.69
	S-2-1	Reverse congestion	$3.37 \cdot 10^{-3}$	0	0	2	791
	S-2-2	Reverse congestion	$3.44 \cdot 10^{-3}$	0	0	2	48.95
	S-2-3	Reverse congestion	$7.26 \cdot 10^{-4}$	0	0	2	50.81
	S-2-4	Reverse congestion	$1.16 \cdot 10^{-2}$	0	0	3	94.20
	S-2-5	Reverse congestion	$7.11 \cdot 10^{-4}$	0	0	2	53.97
	S-3-1	Reverse congestion	$1.99 \cdot 10^{-4}$	0	0	12	604.70
	S-3-2	Length	$2.41 \cdot 10^{-4}$	0	0	1	16.96
	S-3-3	Length	†0	0	0	2	88.69
	S-3-4	Congestion	$1.28 \cdot 10^{-2}$	0	0	7	95.06
	S-3-5	Congestion	$4.89 \cdot 10^{-2}$	0	0	4	121.25
	N-1-1	Random	$2.24 \cdot 10^{-4}$	0	0	5	168.34
	N-1-2	Length	$1.13 \cdot 10^{-3}$	0	0	4	106.44
	N-1-3	Congestion	$4.60 \cdot 10^{-4}$	0	0	3	82.44
	N-1-4	Random	$1.16 \cdot 10^{-3}$	0	0	4	171.13
	N-1-5	Congestion	$4.85 \cdot 10^{-4}$	0	0	5	96.89
	N-2-1	Random	$2.22 \cdot 10^{-3}$	0	0	2	69.46
	N-2-2	Length	†0	0	0	4	97.16
	N-2-3	Length	$1.82 \cdot 10^{-3}$	0	0	3	67.38
	N-2-4	Random	0	0	0	4	153.92
	N-2-5	Length	$\dagger 9.13 \cdot 10^{-4}$	0	0	7	143.64
	N-3-1	Reverse congestion	$2.11 \cdot 10^{-2}$	0	0	13	690.42
	N-3-2	Reverse congestion	$1.95 \cdot 10^{-4}$	0	0	1	17.13
	N-3-3	Reverse congestion	†0	0	0	2	106.09
	N-3-4	Congestion	$8.59 \cdot 10^{-2}$	0	0	10	162.78
	N-3-5	Random	$3.64 \cdot 10^{-4}$	0	0	4	119.06
	Overall		$9.92 \cdot 10^{-3}$	0	0	4.37	138.55

(*continued on next page*)

Table 11 (*continued*)

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
Tabu 10s progressive-2	S-1-1	Reverse congestion	$5.62 \cdot 10^{-2}$	0	0	5	205.71
	S-1-2	Reverse congestion	†0	0	0	4	109.23
	S-1-3	Reverse congestion	$3.32 \cdot 10^{-2}$	0	0	5	78.11
	S-1-4	Congestion	$2.38 \cdot 10^{-2}$	0	0	5	185.61
	S-1-5	Reverse speed	†0	0	0	4	84.69
	S-2-1	Reverse speed	$3.37 \cdot 10^{-3}$	0	0	2	70.91
	S-2-2	Congestion	$3.44 \cdot 10^{-3}$	0	0	2	48.95
	S-2-3	Congestion	$7.26 \cdot 10^{-4}$	0	0	2	50.81
	S-2-4	Congestion	$9.02 \cdot 10^{-4}$	0	0	3	78.43
	S-2-5	Congestion	$7.11 \cdot 10^{-4}$	0	0	2	53.97
	S-3-1	Reverse speed	$1.99 \cdot 10^{-4}$	0	0	12	604.70
	S-3-2	Congestion	$2.41 \cdot 10^{-4}$	0	0	1	16.96
	S-3-3	Congestion	†0	0	0	2	117.50
	S-3-4	Congestion	$1.28 \cdot 10^{-2}$	0	0	7	95.06
	S-3-5	Congestion	†0	0	0	3	109.17
	N-1-1	Reverse congestion	$2.24 \cdot 10^{-4}$	0	0	5	168.34
	N-1-2	Reverse congestion	$1.13 \cdot 10^{-3}$	0	0	4	106.44
	N-1-3	Reverse congestion	$4.60 \cdot 10^{-4}$	0	0	3	82.44
	N-1-4	Conflict time	$1.16 \cdot 10^{-3}$	0	0	4	171.13
	N-1-5	Conflict time	$4.85 \cdot 10^{-4}$	0	0	5	96.89
	N-2-1	Conflict time	$2.22 \cdot 10^{-3}$	0	0	2	69.46
	N-2-2	Conflict time	†0	0	0	4	97.16
	N-2-3	Conflict time	$1.82 \cdot 10^{-3}$	0	0	3	67.38
	N-2-4	Conflict time	†0	0	0	4	153.92
	N-2-5	Conflict time	$9.13 \cdot 10^{-4}$	0	0	7	143.64
	N-3-1	Reverse speed	$4.53 \cdot 10^{-4}$	0	0	12	657.50
	N-3-2	Conflict time	$1.95 \cdot 10^{-4}$	0	0	1	17.13
	N-3-3	Conflict time	†0	0	0	2	106.09
	N-3-4	Reverse speed	$3.47 \cdot 10^{-2}$	0	0	8	116.63
	N-3-5	Conflict time	$3.64 \cdot 10^{-4}$	0	0	4	119.06
	Overall		$5.99 \cdot 10^{-3}$	0	0	4.23	136.10

Table 12
Parallel algorithm results for the RVNS solver on the RAS-based instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
RVNS 2s linear-2	S-1-1	Reverse speed	$5.51 \cdot 10^{-2}$	0	0	5	204.39
	S-1-2	Conflict time	$5.37 \cdot 10^{-2}$	0	0	4	117.14
	S-1-3	Reverse speed	$\dagger 2.04 \cdot 10^{-2}$	0	0	5	73.57
	S-1-4	Reverse speed	$3.69 \cdot 10^{-2}$	0	0	5	190.15
	S-1-5	Conflict time	$1.28 \cdot 10^{-2}$	0	0	4	88.57
	S-2-1	Conflict time	†0	0	0	2	70.05
	S-2-2	Conflict time	†0	0	0	2	47.51
	S-2-3	Conflict time	$7.26 \cdot 10^{-4}$	0	0	2	50.81
	S-2-4	Conflict time	$9.02 \cdot 10^{-4}$	0	0	3	78.43
	S-2-5	Conflict time	$1.42 \cdot 10^{-4}$	0	0	2	53.40
	S-3-1	Reverse speed	†0	0	0	12	604.23
	S-3-2	Conflict time	$7.22 \cdot 10^{-4}$	0	0	1	17.20
	S-3-3	Conflict time	†0	0	0	2	117.50
	S-3-4	Conflict time	$3.57 \cdot 10^{-2}$	0	0	8	106.61
	S-3-5	Conflict time	$1.17 \cdot 10^{-3}$	0	0	3	108.69
	N-1-1	Conflict time	$5.81 \cdot 10^{-1}$	0	0	3	314.07
	N-1-2	Congestion	†0	0	0	4	106.27
	N-1-3	Conflict time	$5.06 \cdot 10^{-3}$	0	0	3	84.08
	N-1-4	Conflict time	$1.45 \cdot 10^{-3}$	0	0	4	171.13
	N-1-5	Conflict time	$5.10 \cdot 10^{-3}$	0	0	5	98.41
	N-2-1	Conflict time	$1.78 \cdot 10^{-3}$	0	0	2	69.36
	N-2-2	Conflict time	$6.33 \cdot 10^{-4}$	0	0	4	97.68
	N-2-3	Conflict time	$3.03 \cdot 10^{-4}$	0	0	3	66.86
	N-2-4	Conflict time	$3.83 \cdot 10^{-4}$	0	0	4	154.45
	N-2-5	Congestion	$6.64 \cdot 10^{-4}$	0	0	7	143.75

(*continued on next page*)

Table 12 (*continued*)

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
	N-3-1	Reverse speed	†0	0	0	12	657.15
	N-3-2	Conflict time	$1.95 \cdot 10^{-4}$	0	0	1	17.13
	N-3-3	Conflict time	†0	0	0	2	106.09
	N-3-4	Conflict time	$4.53 \cdot 10^{-2}$	0	0	7	125.96
	N-3-5	Reverse speed	$9.09 \cdot 10^{-4}$	0	0	4	119.21
	Overall		$2.87 \cdot 10^{-2}$	0	0	4.16	141.99
RVNS 10s progressive-3	S-1-1	Congestion	†0	0	0	5	185.71
	S-1-2	Length	$2.90 \cdot 10^{-2}$	0	0	4	113.67
	S-1-3	Speed	$3.22 \cdot 10^{-2}$	0	0	5	77.76
	S-1-4	Length	$\dagger 2.05 \cdot 10^{-2}$	0	0	5	184.59
	S-1-5	Speed	$3.02 \cdot 10^{-4}$	0	0	4	84.74
	S-2-1	Speed	$3.37 \cdot 10^{-3}$	0	0	2	70.91
	S-2-2	Speed	†0	0	0	2	47.51
	S-2-3	Speed	†0	0	0	2	50.24
	S-2-4	Speed	†0	0	0	3	77.10
	S-2-5	Speed	†0	0	0	2	53.25
	S-3-1	Length	$1.66 \cdot 10^{-4}$	0	0	12	604.58
	S-3-2	Speed	†0	0	0	1	16.96
	S-3-3	Speed	†0	0	0	2	117.50
	S-3-4	Congestion	†0	0	0	8	85.42
	S-3-5	Congestion	$2.35 \cdot 10^{-4}$	0	0	3	108.69
	N-1-1	Congestion	†0	0	0	5	168.25
	N-1-2	Congestion	$1.13 \cdot 10^{-3}$	0	0	4	106.44
	N-1-3	Congestion	†0	0	0	3	82.27
	N-1-4	Length	†0	0	0	4	170.62
	N-1-5	Length	†0	0	0	5	96.72
	N-2-1	Random	†0	0	0	2	68.94
	N-2-2	Length	$5.06 \cdot 10^{-4}$	0	0	4	97.58
	N-2-3	Length	†0	0	0	3	66.76
	N-2-4	Length	$1.53 \cdot 10^{-4}$	0	0	4	154.13
	N-2-5	Length	†0	0	0	7	142.91
	N-3-1	Length	$2.34 \cdot 10^{-2}$	0	0	12	697.16
	N-3-2	Speed	†0	0	0	1	17.08
	N-3-3	Speed	†0	0	0	2	106.14
	N-3-4	Speed	$\dagger 1.36 \cdot 10^{-2}$	0	0	7	98.21
	N-3-5	Speed	†0	0	0	4	118.86
	Overall		$4.15 \cdot 10^{-3}$	0	0	4.23	135.69

References

Acuna-Agost, R., Michelon, P., Feillet, D., Gueye, S., 2011. A mip-based local search method for the railway rescheduling problem. Networks 57 (1), 69–86.

Aly, M., 2015. Survey on multi-class classification methods. Technical Report. Caltech.

Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., Namyst, R., 2010. hwloc: A generic framework for managing hardware affinities in hpc applications. In: Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on. IEEE, pp. 180–186.

Cacchiani, V., Caprara, A., Toth, P., 2010. Scheduling extra freight trains on railway networks. Transp. Res. Part B 44 (2), 215–231.

Cacchiani, V., Huisman, D., Kidd, M., Kroon, L., Toth, P., Veelenturf, L., Wagenaar, J., 2014. An overview of recovery models and algorithms for real-time railway rescheduling. Transp. Res. Part B 63, 15–37.

Caimi, G., Laumanns, M., Schüpbach, K., Wörner, S., Fuchsberger, M., 2011. The periodic service intention as a conceptual framework for generating timetables with partial periodicity. Transp. Plan. Technol. 34 (4), 323–339.

Caprara, A., Fischetti, M., Toth, P., 2002. Modeling and solving the train timetabling problem. Oper. Res. 50 (5), 851–861.

Clausen, J., Perregaard, M., 1999. On the best search strategy in parallel branch-and-bound: best-first search versus lazy depth-first search. Ann. Oper. Res. 90, 1–17.

Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2009. Evaluation of green wave policy in real-time railway traffic management. Transp. Res. Part C 17 (6), 607–616.

Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2010a. Centralized versus distributed systems to reschedule trains in two dispatching areas. Public Transp. 2 (3), 219–247.

Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2010b. A tabu search algorithm for rerouting trains during rail operations. Transp. Res. Part B 44 (1), 175–192.

Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2012. Bi-objective conflict detection and resolution in railway traffic management. Transp. Res. Part C 20 (1), 79–94.

Corman, F., D'Ariano, A., Pranzo, M., Hansen, I.A., 2011. Effectiveness of dynamic reordering and rerouting of trains in a complicated and densely occupied station area. Transp. Plan. Technol. 34 (4), 341–362.

Corman, F., D'Ariano, A., Marra, A.D., Pacciarelli, D., Samà, M., 2016. Integrating train scheduling and delay management in real-time railway traffic control. Transp. Res. Part E.

Corman, F., Meng, L., 2015. A review of online dynamic models and algorithms for railway traffic management. IEEE Trans. Intell. Transp. Syst. 16 (3), 1274–1284.

D'Ariano, A., Corman, F., Pacciarelli, D., Pranzo, M., 2008a. Reordering and local rerouting strategies to manage train traffic in real time. Transp. Sci. 42 (4), 405–419.

D'Ariano, A., Pacciarelli, D., Pranzo, M., 2007. A branch and bound algorithm for scheduling trains in a railway network. Eur.J. Oper. Res. 183 (2), 643–657.

D'Ariano, A., Pacciarelli, D., Pranzo, M., 2008b. Assessment of flexible timetables in real-time traffic management of a railway bottleneck. Transp. Res. Part C 16 (2), 232–245.
D'Ariano, A., Pranzo, M., 2009. An advanced real-time train dispatching system for minimizing the propagation of delays in a dispatching area under severe disturbances. Netw. Spatial Econ. 9 (1), 63–84.
Dollevoet, T., Huisman, D., Kroon, L., Schmidt, M., Schöbel, A., 2014. Delay management including capacities of stations. Transp. Sci. 49 (2), 185–203.
Fang, W., Yang, S., Yao, X., 2015. A survey on problem models and solution approaches to rescheduling in railway networks. IEEE Trans. Intell. Transp. Syst. 16 (6), 2997–3016.
Hansen, I., Pachl, J., 2014. Railway Timetabling & Operations. Eurailpress, Hamburg.
Hansen, P., Mladenović, N., Brimberg, J., Pérez, J., 2010. Variable neighborhood search. In: Gendreau, M., Potvin, J.-Y. (Eds.), Handbook of Metaheuristics. In: International Series in Operations Research & Management Science, 146. Springer, pp. 61–86.
INFORMS Railways Applications Section, 2012. Problem description and released data set for the ras problem solving competition. URL https://www.informs.org/Community/RAS/Problem-Solving-Competition/2012-RAS-Problem-Solving-Competition.
Iqbal, S.M.Z., Grahn, H., Krasemann, J.T., 2013. Multi-strategy based train re-scheduling during railway traffic disturbances. In: Proceedings of the 5th International Seminar on Rail Operations Modeling and Analysis (RailCopenhagen 2013, pp. 387–405), Technical University of Denmark, Denmark.
Iqbal, S.M.Z., Grahn, H., Krasemann, T., et al., 2012. A parallel heuristic for fast train dispatching during railway traffic disturbances: Early results. In: 1st International Conference on Operations Research and Enterprise Systems, ICORES.
Kanai, S., Shiina, K., Harada, S., Tomii, N., 2011. An optimal delay management algorithm from passengers' viewpoints considering the whole railway network. J. Rail Transp. Plan. Manag. 1 (1), 25–37.
Lamorgese, L., Mannino, C., 2013. The track formulation for the train dispatching problem. Electron. Notes Discrete Math. 41, 559–566.
Lamorgese, L., Mannino, C., 2015. An exact decomposition approach for the real-time train dispatching problem. Oper. Res. 63 (1), 48–64.
Li, F., Gao, Z., Li, K., Yang, L., 2008. Efficient scheduling of railway traffic based on global information of train. Transp. Res. Part B 42 (10), 1008–1030.
Mascis, A., Pacciarelli, D., 2002. Job-shop scheduling with blocking and no-wait constraints. Eur. J. Oper. Res. 143 (3), 498–517.
Meng, L., Zhou, X., 2011. Robust single-track train dispatching model under a dynamic and stochastic environment: a scenario-based rolling horizon solution approach. Transp. Res. Part B 45 (7), 1080–1102.
Meng, L., Zhou, X., 2014. Simultaneous train rerouting and rescheduling on an n-track network: a model reformulation with network-based cumulative flow variables. Transp. Res. Part B 67, 208–234.
Mu, S., Dessouky, M., 2011. Scheduling freight trains traveling on complex networks. Transp. Res. Part B 45 (7), 1103–1123.
Pellegrini, P., Marlière, G., Rodriguez, J., 2014. Optimal train routing and scheduling for managing traffic perturbations in complex junctions. Transp. Res. Part B 59, 58–80.
Rodriguez, J., 2007. A constraint programming model for real-time train scheduling at junctions. Transp. Res. Part B 41 (2), 231–245.
Ropke, S., Santini, A., 2016. Parallel adaptive large neighbourhood search. Technical Report OR-16-11. DEI University of Bologna.
Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. Oper. Res. 177 (3), 2033–2049.
Samà, M., Pellegrini, P., D'Ariano, A., Rodriguez, J., Pacciarelli, D., 2016. Ant colony optimization for the real-time train routing selection problem. Transp. Res. Part B 85, 89–108.
Santini, A., 2017. cr-ras-derived-instances: Initial release. https://doi.org/10.5281/zenodo.322571.
Schachtebeck, M., Schöbel, A., 2010. To wait or not to wait—and who goes first? delay management with priority decisions. Transp. Sci. 44 (3), 307–321.
Schöbel, A., 2007. Integer programming approaches for solving the delay management problem. In: Geraets, F., Kroon, L., Schoebel, A., Wagner, D., Zaroliagis, C. (Eds.), Algorithmic Methods for Railway Optimization. In: Lecture Notes in Computer Science, 4359. Springer Berlin Heidelberg, pp. 145–170.
Schöbel, A., 2009. Capacity constraints in delay management. Public Transp. 1 (2), 135–154.
Törnquist, J., 2012. Design of an effective algorithm for fast response to the re-scheduling of railway traffic during disturbances. Transp. Res. Part C 20 (1), 62–78.
Törnquist, J., Persson, J., 2007a. N-Tracked railway traffic re-scheduling during disturbances. Transp. Res. Part B 41 (3), 342–362.
Törnquist, J., Persson, J.A., 2007b. N-Tracked railway traffic re-scheduling during disturbances. Transp. Res. Part B 41 (3), 342–362.