The Tick Formulation for Deadlock Detection and Avoidance in railways traffic control[⋆,⋆⋆]

Veronica Dal Sasso[a,*], Leonardo Lamorgese[a], Carlo Mannino[b], Andrea Onofri[a] and Paolo Ventura[c]

[a] *Optrail, viale Marco Polo 59, 00154 Rome, Italy*
[b] *SINTEF Digital and University of Oslo, Norway*
[c] *IASI-CNR, Rome, Italy*

ARTICLE INFO

Keywords:
deadlock detection
railways networks
linear integer programming
Tick formulation

ABSTRACT

Wrong dispatching decisions may lead to *deadlocks*, where trains reciprocally block resources necessary to reach their destinations. It is crucial to develop tools to detect such potential deadlocks on time, in order to reverse the decisions previously taken by dispatchers or to take recovery actions. In this paper we present a new 0,1 linear formulation for detecting deadlocks and optimally park the involved trains to reduce congestion around the affected area. We discuss computational results on some realistic randomly generated instances to show the validity of the approach, as well as its limits.

1. Introduction

Railway transportation management gives rise to a wide range of challenging problems. Official timetables are built to assign where and when a train should be routed from its origin towards its destination, and they are executed under the supervision of *dispatchers*. It is not uncommon that unforeseen and disrupting events happen, thus delaying trains or, in the worst cases, making the current timetable infeasible. Hence, one of the dispatchers' tasks consists of taking recovery actions to ensure feasibility of the train schedule. This is done, in particular in recent years, with the aid of dispatching tools and dedicated software (See, for example, Bollapragada, Markley, Morgan, Telatar, Wills, Samuels, Bieringer, Garbiras, Orrigo, Ehlers et al. (2018); Aurizon (2020)). The purpose of these softwares is to help dispatchers to avoid *deadlocks*, situations in which a subset of trains is blocking resources mutually needed and no train can move forward anymore. Deadlocks are usually caused by decisions, taken by dispatchers, whose consequences are not easy to spot sufficiently in advance. Once one of such "fatal" decisions is implemented, for instance routing a train on a specific track, then a certain set of trains will irreversibly end up in a deadlock as they proceed towards destination. Note that such unavoidable deadlocks in general appear only after some or all trains have moved forward: in this case we say that the trains are *bound-to-deadlock*.

Pachl (2011) provides a very in-depth introduction to deadlocks in railway systems. Pachl also lists three strategies to handle deadlocks or bound-to-deadlock configurations in computer, telecommunication and transport systems. Two out of these three strategies, namely *deadlock detection*, and *deadlock avoidance*, are considered in this paper. Deadlock avoidance aims at never allowing a train to move in a way which would bring the system into a bound-to-deadlock configuration. Deadlock detection amounts instead to identifying a bound-to-deadlock configuration. In principle, taking Pachl's deadlock avoidance definition literally, we would need to decide whether a candidate movement of a train leads to a bound-to-deadlock configuration: this is a *decision problem* (Garey and Johnson (1979)), which requires a deadlock detection test. Pachl and other authors instead choose a different, heuristic approach to deadlock avoidance. Somehow paradoxically, Pachl regards deadlock detection not to be of interest for railway systems, because recovery actions are difficult to implement when trains are involved.

Nonetheless, we are interested in deadlock detection because deadlocks, although rare, *do occur* in practical dispatching. In particular, they are more frequent in predominantly freight-based railways (such as North-America) where

⋆ This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.
⋆⋆ Declarations of interest: none
veronica.dalsasso@optrail.com (V. Dal Sasso); leonardo.lamorgese@optrail.com (L. Lamorgese); carlo.mannino@sintef.no (C. Mannino); andrea.onofri@optrail .com (A. Onofri); paolo.ventura@iasi.cnr.it (P. Ventura)
ORCID(s):

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

rail companies operate without a schedule (timetable) and trains can be up to five kilometers long. Recovery actions then result in large economic loss and significant impact on train punctuality. Recovery from a bound-to-deadlock configuration involves pulling back at least one train (since moving forward all involved trains leads to a certain deadlock). This, in turn, requires dedicated personnel and one or more service trains to be sent to the train(s) target of the push back. To this end, other trains between the origin of the service train and the target trains must be parked in a position (a "*safe place*") which allows the service train to reach the target train.

The approach developed in this paper is designed to be the planning core of a sub-system of an ATMS (Autonomous Traffic Management System). An ATMS is an on-line software system that produces routing and scheduling plans for a "rolling" time horizon to optimize train traffic. These plans are continuously recalculated and can be provided as "decision-support" to dispatchers or carried out "directly" by the ATMS. A dispatching decision, e.g. routing a train to a specific track, is implemented by sending the associated commands to the field, where in turn signals will be lighted and switches activated accordingly. In principle, plans suggested by the Planner should always lead to deadlock-free situations. However, in whichever mode the system operates (decision-support or fully autonomous), dispatchers have the ability to introduce commands to take (or override) decisions. This leads to the chance of a specific command (or set of commands) creating a bound-to-deadlock configuration. A deadlock detection algorithm described in this paper can thus be embedded in a component that comes into play in this context. If the algorithm detects a bound-to-deadlock configuration when a dispatching decision is communicated to the system, the system can either invalidate these commands or send a warning message to the dispatcher to abort the decision (deadlock avoidance). On the other hand, if due to timing reasons the dispatching decision is already implemented and some trains are now bound-to-deadlock, still early detection is very relevant. Indeed, before applying recovery actions, a safe place must be found for the approaching trains that are not to be pushed back: the earlier the bound-to-deadlock situation is detected, the easier it is to identify good safe places. In these cases, the algorithm's safe place assignment becomes an input for the ATMS' scheduling and routing algorithm, that treats these locations as destinations for the involved trains in the following planning cycles, until the deadlock is lifted.

Trains run through the network following a given trajectory, that is a path in the network and a schedule specifying when a train is in each element of its path. The set of paths assigned to the trains is often denoted as *routing*. Deadlocks occur because distinct trains share network resources. The paths and schedules - i.e. the *plan* - of the controlled trains are defined to avoid *conflicts*, that is that two trains are scheduled to run the same track at the same time[1]. In these cases, conflicts may be resolved by allowing local re-routings, i.e. changing platforms or tracks used, or re-routings which also modify the path of the trains. Note that, if we can find a conflict-free plan, then the trains are not bound-to-deadlock. On the other hand, if the trains are not bound-to-deadlock, then such a conflict-free plan exists. Actually, this equivalence provides us with a bit more precise definition[2]:

Definition 1. A set of trains is bound-to-deadlock if and only if it is not possible to find a conflict-free plan that routes the trains to their destinations.

The problem of checking if a given train configuration is bound-to-deadlock is known to be NP-complete[3] (Lu, Dessouky and Leachman (2004)).

In the literature, deadlock avoidance is actually pursued by explicitly or implicitly looking for feasible plans (e.g. in Cui (2009); Cui, Martin and Liang (2017); Li, Sheu and Gao (2014); Lu et al. (2004); Pachl (2011)). An implicit plan is given by an origin-to-destination path for each train (*routing*) plus the ordering in which trains visit the shared resources. If such order is acyclic, then a schedule can be immediately and efficiently derived by computing a longest path tree in an acyclic directed graph (Ahuja, Magnanti and Orlin (1993)).

Banker's algorithm applied to railway scheduling, as described for instance in Cui et al. (2017), is a simple and quick method to look for one such feasible plan. The idea is trivial. Assume the routing is fixed. Suppose you find a train which can run from origin to destination with no conflicts with other trains while holding all the other $n-1$ trains at their origins. Then you can safely schedule this train while holding the other trains. Now you still have to plan $n-1$ trains. Still, you may find a second train which can run safe from origin to destination with no conflicts with the remaining $n-2$ trains hold at their respective origins. By recursively applying this principle, if you are lucky, you will find a full plan. Clearly, a conflict-free schedule may exist even if Banker's algorithm does not find it, and this case is

[1] Conflicts may occur also when trains occupy different tracks, but this is not relevant here.

[2] In this definition we are deliberately ignoring many details, however not needed in the discussion that follows.

[3] For a more general discussion on the complexity of deadlock in job-shop scheduling, see also Arbib, Italiano and Panconesi (1990).

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

called a *false-positive*, because one may be tempted to wrongly conclude that trains are bound-to-deadlock. Moreover, the quality of the plans associated to Banker's algorithm can be very poor (see Lu et al. (2004)). Banker's algorithm is indeed so naive that in most practical situations it will end up with a false-positive. Cui (2009) and Cui et al. (2017) discuss different heuristic procedure to improve Banker's algorithm in order to reduce the number of false-positives. A similar path is followed in Lu et al. (2004), where Banker's algorithm (Algorithm 4.1. in the paper) is improved to find feasible schedules, therefore a deadlock free solution. Also Pachl (2011) present a heuristic approach to deadlock avoidance, based on rules. It returns an implicit schedule by providing the order in which trains visit each track. Even though Li et al. (2014) presents an exact MILP model to the scheduling problem which in principle could detect bound-to-deadlock configurations, ultimately they resort to a heuristic scheduling algorithm, although equipped with some smart deadlock detection sub-modules. Summarizing, the methods for deadlock avoidance described in Cui et al. (2017); Li et al. (2014); Lu et al. (2004); Pachl (2011) are indeed heuristics to find feasible schedules; therefore, they cannot answer "YES" to the question if a set of trains is bound-to-deadlock. Moreover, they do not consider re-routing of trains, which may be necessary for avoiding deadlocks.

Re-routing is also not allowed in the instances tackled in Mazzanti, Spagnolo, Della Longa and Ferrari (2011). However, the approach is exact, false positives cannot occur and, if the plan is not found, then one can indeed conclude that the trains are bound-to-deadlock. The idea is to build a verifiable formal model of the system and then apply formal model checking.

Actually, thanks to the equivalence of deadlock avoidance and conflict-free planning, there is a vast literature on railway planning from which we can get inspiration. In principle, any heuristic algorithm for planning trains can be used to heuristically avoid deadlock, and any exact algorithm can also be used to detect bound-to-deadlock configurations. The literature on this topic is so vast that we must refer the reader to recent surveys such as Fang, Yang and Yao (2015); Wen, Huang, Li, Lessan, Fu, Jiang and Xu (2019). Note that the methods explicitly designed to plan trains, rather than to avoid deadlocks, have the additional advantage that they try to return good (in terms of delay minimization) plans, not simply feasible plans. Also, due to the well known observation that planning trains is a job-shop scheduling problem with some extra constraints (Mascis and Pacciarelli (2002)), one can actually draw from the gigantic body of scientific literature on machine scheduling. There exist a few mathematical models for job-shop scheduling problems. For planning trains in real-time, big-M formulations (e.g. (Balas (1985)) are most often adopted. The second alternative are the so called time-indexed formulations (Dyer and L. (1990); Queyranne and Schulz (1994)), mainly adopted for producing off-line timetables (as in Harrod (2011)). Our approach can be seen as a variant of the latter.

Time-indexed formulations require the time horizon to be discretized into time periods and the main variables, describing the positions of trains over time, are indexed over each time period. The main difficulty of these formulations lies in the choice of the discretization step: too large a step may generate plans which cannot be implemented in practice (see, for instance, Harrod (2011)), whereas small steps result in a very high number of integer variables and a substantial increase of computational times. This is why time-indexed formulations are preferred for off-line planning problems, like train timetabling (Caprara, Fischetti and Toth (2002); Harrod (2011)) and track assignment . With few exceptions devoted to train planning, like Caimi, Fuchsberger, Laumanns and Lüthi (2012); Meng and Zhou (2014); Reynolds, Ehrgott, Maher, Patman and Wang (2020), which however must resort to various schemes in order to limit the number of variables.

In this paper we introduce a new pure 0,1 programming formulation, and call it *Tick Formulation*, to model the Deadlock Detection (DD) problem. It is based on the idea that train movements can be decomposed into sequences of shorter movements or *moves* and that moves for different trains may be performed simultaneously, or during the same *tick*. Rather than having time-indexed variables, we will have tick-indexed variables, specifying in which tick a move is performed. If $j > i$, the moves (train movements) performed at tick j will occur after all the moves (train movements) performed at tick i. In this, the approach recalls the event-based model for project scheduling described in Koné, Artigues, Lopez and Mongeau (2011); Tesch (2020). However, "events" are replaced by moves and there are no continuous variables to represent schedules, since the precise timing is not relevant for the determination of deadlocks. Indeed, deadlocks happen when trains cannot be routed to their destination, and one can show that this is only due to the order in which they traverse the shared tracks. Our Tick Formulation may resemble the positional formulation for single machine scheduling problems (see Keha, Khowala and Fowler (2009)) extended to multiple machines and with no scheduling variables.

Another contribution of this paper is to extend the Tick Formulation to answer the question: *which are the bound-to-deadlock trains?* In fact, we can detect a subset of trains responsible for the deadlock (the *culprit* set), such that removing some trains in the set would lift the deadlock.

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

Next section defines the problem and the notation used throughout the paper, while Section 3 describes the model. Section 4 proves the validity of the model. In Section 5 we analyse the results obtained on a benchmark of randomly generated instances. Finally, Section 6 concludes the paper and anticipates future developments.

2. The Deadlock Detection problem: definitions and notation

In this section we introduce the mathematical formalism to describe the railway network and train movements across the network, and we give a formal statement of the problem. Note that the target of this study is to develop a methodology which can be used in the practice by dispatchers to avoid or detect deadlocks in real-time. The focus is on North American freight railway systems, because, also according to Pachl (2011), these are prone to end up in deadlock configurations since "there is no scheduling process that prevents the traffic management from putting too many trains on a line". In Pachl (2011) it is also observed that, thanks to the low traffic density of North-American freight railways, the chances of having deadlocks caused by more than three trains are negligible. To be a bit more formal, consider a configuration of M trains running in the network, and assume that such configuration is bound-to-deadlock. We say that the deadlock is caused by a (culprit) set $C \subseteq M$ of trains when if we remove from the network all trains $m \in M \setminus C$, the system of the C trains is still bound-to-deadlock, but removing one more train (from C) would lift the deadlock. So C is a minimal set of trains which is bound-to-deadlock. If trains are bound-to-deadlock, then one or more such minimal sets C exist. Now we can rewrite Pachl observation by stating that, in the practical North-American context, a culprit set C, with high probability, has cardinality $|C| \leq 3$. Note that, if C is a culprit set, then at least one train in C must be "removed" so to lift the deadlock configuration. In a practical context, removing a train amounts to pushing it back to some station or siding (any forward movement would still result in a bound-to-deadlock configuration). Such a train must thus be identified, and then a service train may be necessary to perform the recovery. As mentioned, the overall procedure is very costly and time consuming.

Hence, we can define the Deadlock Detection Problem as follows:

Definition 2. Let M be a set of trains running on a railway network. Given the positions of the trains at a particular time, we want to identify if a bound-to-deadlock situation is in place and, if yes, which trains are the ones bound-to-deadlock.

Actually, in what follows we will focus on a suitable portion[4] of the overall railway network which will be represented by a directed graph (we will give the formal definition of such graph later in the section). In the graph representation we can identify a set of nodes as *boundary nodes*, that is all trains entering or exiting our portion of network must go through one of these nodes. In our model we assume that boundary nodes have infinite capacity and can accommodate any number of trains, as they represent in some sense the external portion of network.

Rather than identifying one or more culprit sets, our method will find a tripartition Q, C, P of the set M of trains such that: (i) the trains in Q can still continue running to their destination; (ii) the trains in C will be halted in their current position, waiting for recovery actions to solve the deadlock; (iii) the trains P cannot reach their destination because of the trains in C, but they should not be considered responsible for the deadlock as they can be parked in suitable *safe places*. In particular, we will use the following definition of safe place parking.

Definition 3. The trains of P are in a safe place if a generic train $m \notin M$ can enter the network from any boundary node and leave it from any other boundary node without using the tracks occupied by the trains of P.[5]

We remark thus that the problem tackled in this paper is *an extension of the Deadlock Detection Problem*. Indeed, our scope is to answer the yes/no question asking whether a bound-to-deadlock situation is in place and, if so, to find the subset of trains that should be held or parked at suitable safe places.

Figures 1 and 2 give a graphical representation of some definitions introduced so far. Figure 1 shows an example of bound-to-deadlock configuration (on the left). In this case, the trains are still far away, but they are too long to find a station in between where they can be routed past one another. We aim at identifying such situation before actually reaching the deadlock depicted on the right of Figure 1, as the sooner we identify the culprit set of trains, the sooner the recovery actions could take place. Figure 2 shows an example of a train (the darkest) parked at a safe place.

[4] In practice, such portion must be identified by pre-processing the current dispatching instance, along with a candidate set M of involved trains.
[5] Moreover, the set C we will provide is the smallest possible.

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

Figure 1: Trains bound-to-deadlock (left) and in a deadlock (right)

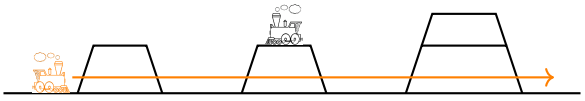

Figure 2: Black train parked at a safe place

The model discussed in this paper is very general, and can be applied to any whatsoever complex network. However, for ease of explanation, we will limit our discussion to *single-line* networks, that is networks where there is only one track between adjacent stations, which then must be used for trains running in both directions. The extension of the model to multi-track networks (i.e., networks where stations are connected by more than one distinct track, which can be used in both directions), terminal stations, or other complex layouts, is straightforward and would not add much to the discussion. Moreover, as discussed above, deadlocks are much more frequent in *single-line* regions, making this case particularly relevant in practical contexts.

The physical railway network is made of directed *routes*[6], starting from a physical signal and ending at the following signal in the same direction. See, for example, Figure 3: signals 1 and 3 are the extremes of one eastwards route. Each route is made up of a *stopping point*, that is the segment of the route before the ending signal (which must not interfere with any switch) and an *entering track*, which is the segment of the route that leads to the stopping point (which may include switches). A typical example of a stopping point is a station platform, where passengers can board or alight trains (e.g segment $2-3$ in Figure 3, with segment $1-2$ being the entering track). Similarly, a open line track can also be a stopping point (e.g. segment $4-5$).

In the sequel, we make the following **assumptions:**

1. *at any point in time, there can be at most one train per route,*
2. *any train can stop at any stopping point.*

Assumption 1 holds as we are in a single-line scenario, but it needs to be dropped when extending the formulation to multi-track networks. In fact, it may be necessary to have more than one train on the same route at the same moment to allow trains in the opposite direction to bypass them towards their destination, thus avoiding to create deadlocks. For this reason, multi-track scenarios require further attention on the tracks capacity and the order in which trains enter and exit tracks. By Assumption 2, we can assume a train can stop with its head at the end of any route, waiting for its conflict-free path. This assumption holds also for more complex networks.

Notice that two routes in opposite directions, as routes c and d of Figure 3, may share the same stopping point, while a pair of routes, as a and b, may intersect at their entering tracks. Clearly, when one train occupies a stopping point, no other train can travel through it. If, instead, trains are longer than the stopping point, they occupy also the switch, impeding other trains from traversing the network (for example, if a train stops on route a and it is longer than the stopping point, no other train can traverse route b). This leads to two kinds of route incompatibility. Let a be a route:

- a route is *incompatible* with a if it shares the same stopping point (and we denote this set of incompatible resources with $\mathcal{I}_a$);
- a route is *potentially incompatible* with a if it shares a portion of the entering track. The incompatibility becomes effective only when the trains lengths are accounted for and the trains do not fit within the respective stopping points (we denote this set as $\mathcal{I}_a^{sw}$).

As mentioned in the introduction, we want to decompose the train movements into sequences of moves:

[6]We remark here that we are picking the term " route" from the standard railway technical jargon. "Routes" in vehicle routing problems, for instance, denote something different.

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

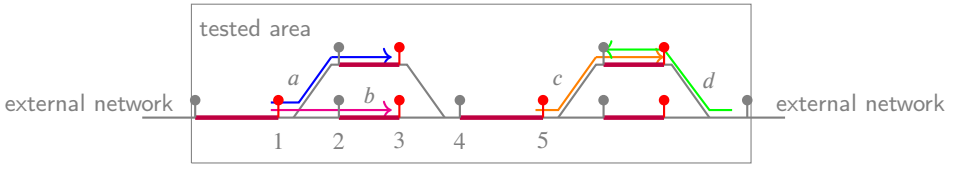

Figure 3: An example of a single-line railway network

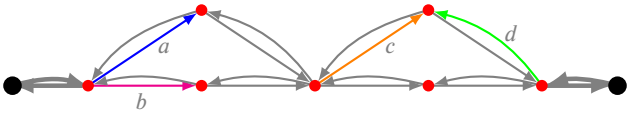

Figure 4: $D = (V, A)$: representations of the railways network in Figure 3 as a directed graph. Nodes represent stopping points, while arcs represent routes (signal to signal).

Definition 4. A *move* is the progress of a train's head from one signal along its path to one of the following signals in the same direction.

A subset of trains can start their next move at the same time. We say that these moves take place at the same tick:

Definition 5. A *tick* represents an (unspecified) continuous time interval in which a subset of trains start and complete a move each.

Hence, with the Tick formulation we want to model, if it exists, a sequence of moves for each train which ensures a deadlock-free plan. At most one of these moves takes place at each tick, but at any tick several trains can make a move. Moreover, a train may stay still for several ticks, if it needs to wait for the railway routes to be freed by other trains. When such a sequence cannot be found for the whole set of trains, it means that there is a bound-to-deadlock configuration. In this case, the Tick formulation will return the aforesaid tripartition, and in particular the subset C of trains that must be stopped at their current position.

We model the railway network as a directed graph $\bar{D} = (\bar{V}, \bar{A})$ (see Figure 4, which represents the network in Figure 3) where the nodes represent stopping points and the arcs represent routes from a stopping point to the next one. The boundary between the tested area and the external network is modeled with an additional set H of boundary nodes (the bigger nodes in Figure 4) which are nodes that belong to the external network but have at least an adjacent node within the tested area. The set of arcs B (the thicker ones) links the boundary nodes to the nodes of the network. The arcs B are called *black holes* and they represent the exit from the portion of network in which we are looking for deadlocks. The number of these boundary nodes and black holes may vary, according to the trains involved: if the timetable of a train ends at a yard that is placed in the middle of the single-tracked line under investigation, a black hole (and its boundary node) is added there to let this train follow its original path without being stuck in the tested area. Let $D = (V, A)$, with $V = \bar{V} \cup H$ and $A = \bar{A} \cup B$, be the final representation of the railway network. From now on, we will refer to network routes as route arcs and to stopping points as stopping nodes. For each $a \in A$, let c_a be the length of route arc a and $\mathcal{I}_a$, $\mathcal{I}_a^{sw}$ its incompatibility sets. Moreover, for each $s \in V$, let w_s be the length of stopping node s. Notice that route arcs belonging to B have infinite length and no incompatibilities.

The trains moving along the network are identified by a set $M = M_L \cup M_R$, where M_L (resp. M_R) are the westward (eastward) trains. For each train $m \in M$ we define:

1. the length l_m,
2. an ordered set $r_m^1, ..., r_m^{q_m}$ of route arcs currently occupied, where r_m^1 hosts the head of the train and the other route arcs host the (if any) tails,
3. a nonempty set $S_m \subseteq \bar{V}$ of feasible safe places,
4. a subset of reachable black holes $B_m \subseteq B$.

While for time-indexed formulations the number of time intervals is trivial to compute, once the discretization step is decided, here we need to determine the maximum number of ticks $[T] = [1, \dots, T]$ that trains can use to reach their destination. Notice that this number takes into account also ticks where trains idle, waiting for other trains. Hence, in

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

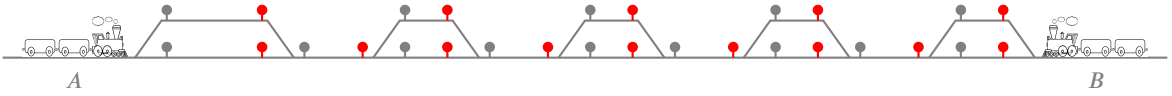

Figure 5: An example of instance where a number of ticks near to the UB is needed

general, the ticks needed by a train are more than its moves. In the most optimistic case, no train has to wait for another train, and the lower bound on T can be set to the maximum number of moves that the farthest train has to do to reach its black hole. On the other hand, the upper bound on T is given by the sum over all trains of their maximum number of moves to reach their farthest black hole, assuming only one train performs a single move at each tick:

$$LB = \max_{m \in M} \max_{b \in B_m} \{\#\text{ moves to reach } b \text{ from the current position of } m\} \quad (1)$$

$$UB = \sum_{m \in M} \max_{b \in B_m} \{\#\text{ moves to reach } b \text{ from the current position of } m\} \quad (2)$$

There is no guarantee of a number of ticks in between that is always enough to find the optimal solution. In extreme cases, as depicted in Figure 5, T needs to be almost equal to the upper bound: both trains need to perform 10 moves, hence we have $LB = 10$ and $UB = 20$. However, train A has to stop at the first station and wait for train B for 8 ticks, so in order to route both trains across the network we need at least 18 ticks. The issue can be easily overcome by considering $T = UB$. However, this can prove to be impractical for realistic scenarios. How the selection of T affects the model performance will be shown in Section 5.

For each $m \in M$, we denote by $A_m \in A$ the subset of route arcs feasible for train m and with $p(m,a) \subseteq A_m$ (resp. $f(m,a)$) the set of route arcs that precede (follow) a, reachable by m. Notice that, as the definition of A_m is very general, we are also allowing re-routings for any train m. Moreover, $\delta^-(s)_m$ denotes the set of route arcs incoming the stopping node s in V that can be traversed by train m.

3. The Tick formulation

This Section describes the Tick Formulation for modelling the DD problem.

In order to choose safe places which adhere to Definition 3, we introduce an extra set $\bar{M}$ of dummy trains of unitary length. A feasible safe place assignment for a subset of trains in M, which can be deduced from the route arcs occupied by trains at tick T, is one that allows dummy trains to traverse the network. Hence, we introduce a dummy train $\bar{m} \in \bar{M}$ for each pair of boundary nodes and their black holes $o_{\bar{m}}, d_{\bar{m}}$. This train originates at a boundary node and starts to traverse the network at the black hole $o_{\bar{m}}$. Then it exits from the network by reaching its black hole $d_{\bar{m}}$. In doing this, it needs to avoid resources that are incompatible with the ones occupied by trains in safe places. Note that, on the contrary, we are not asking that dummy trains avoid trains that need to be stopped at their initial position. Hence, we introduce a set of extra ticks $[\bar{T}] = [T+1, \bar{T}]$ in which only dummy trains can move. Trains belonging to M, instead, will occupy at each tick in $[\bar{T}]$ the same resources they occupy at tick T.

In case there is more than one dummy train, they are allowed also to pass through each other or occupy the same resource at the same time.

We first define the used variables. Constraints describe how the trains move on the network and how dummy trains are handled in order to park the trains at valid safe places. Finally, we define the objective function.

3.1. Variables

Variables record which route arcs are occupied by each train at each tick. Recall that trains in M can move only in the first T ticks, while from tick $T+1$ to $\bar{T}$ only dummy trains can move. Hence, we avoid creating unnecessary variables for trains in M and ticks in $[\bar{T}]$, by remembering that they would assume the same value of the corresponding variable at tick T. We define for each $(m,a,t) \in M \times A \times [T]$ and $(m,a,t) \in \bar{M} \times A \times [\bar{T}]$ variables $x_{m,a,t}$ for the head of the train. If the train is too long to fit in just one route arc, its tail will occupy other routes, that are recorded by variables $y_{m,a,t}$ for each $(m,a,t) \in M \times A \times [T]$ (Notice that dummy trains do not require variables for the tails). We also introduce an extra set of variables h_m for each $m \in M$, which records if a train must be stopped at its initial position:

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

$$x_{m,a,t} = \begin{cases} 1 & \text{if train } m \text{ has its head on route arc } a \text{ at tick } t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$y_{m,a,t} = \begin{cases} 1 & \text{if the tail of train } m \text{ occupies route arc } a \text{ at tick } t \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$h_m = \begin{cases} 1 & \text{if train } m \text{ must be stopped at its initial position} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

3.2. Constraints

The following sets of constraints model how the trains move along the network, from tick 0 to tick T:

1. at the beginning, the current positions of the trains head and tails are known:

$$x_{m,r_m^1,0} = 1, \quad \forall\, m \in M, \tag{6}$$

$$y_{m,a,0} = 1, \quad \forall\, m \in M, \forall\, a \in \{r_m^2, \dots, r_m^{q_m}\} \tag{7}$$

2. at each tick, each train's head is allocated to exactly one of its feasible routes:

$$\sum_{a \in A_m} x_{m,a,t} = 1, \quad \forall\, m \in M, \forall\, t \in [T] \tag{8}$$

3. at each tick, each stopping node s in $\bar{V}$ is occupied by at most one train in M, either by its head or by its tail:

$$\sum_{m \in M} \sum_{a \in \delta^-(s)_m \cap \bar{A}} (x_{m,a,t} + y_{m,a,t}) \leq 1, \quad \forall\, s \in \bar{V}, \forall\, t \in [T] \tag{9}$$

4. at each tick, a train can either stay still or move forward to one of its following route arcs:

$$x_{m,a,t} \leq x_{m,a,t+1} + \sum_{a' \in f(a,m)} x_{m,a',t+1} \quad \forall\, m \in M, \forall\, a \in A_m, \forall\, t \in [T-1] \tag{10}$$

5. the following constraints ensure tails are activated when needed: let $A' \subseteq A_m$, $A' = \{a, a_0, \dots, a_k\}$ be a subset of route arcs occupied by train m (with the head on arc a). Suppose that arcs in A' are not long enough to contain the whole train m, that is, $l_m < c_a + \sum_{i=0}^{k} c_{a_i}$. Then we know that m also occupies at least an extra route arc, hence we want to activate an extra tail and place it on one of the arcs preceeding a_k. Let $\mathcal{A}'$ be the set of all possible sets A':

$$x_{m,a,t} \leq \sum_{i=0}^{k} (1 - y_{m,a_i,t}) + \sum_{a' \in p(a_k,m)} y_{m,a',t}, \quad \forall m \in M, \forall\, A' \in \mathcal{A}' \forall\, t \in [T] \tag{11}$$

Note that, as the route for any train m is not fixed, we write constraints (11) for each subset A' whose routes are not long enough to fit the whole train. As an example, consider Figure 6 and a train m with head on route arc 8 at tick t. As m is long, it does not fit in just one route arc. Hence $A' = \{8\}$ and the corresponding constraint ensures that either $y_{m,5,t} = 1$ or $y_{m,6,t} = 1$. Moreover, let's suppose m does not fit within the route arcs 5 and 8. Hence $A' = \{5, 8\}$ and the corresponding constraint asks that also $y_{m,3,t} = 1$. If, however, route arcs 6 and 8 are long enough to fit the whole train, no constraint will be written for $A' = \{6, 8\}$.

Constraints (12) and (13) ensure continuity in space and time of head and tails of each train.

$$y_{m,a,t} \leq \sum_{a' \in f(m,a)} (x_{m,a',t} + y_{m,a',t}), \qquad \forall\, m \in M, \forall\, a \in A_m, \forall\, t \in [T] \tag{12}$$

$$y_{m,a,t+1} \leq x_{m,a,t} + y_{m,a,t}, \qquad \forall\, m \in M, \forall a \in A_m, \forall\, t \in [T-1] \tag{13}$$

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

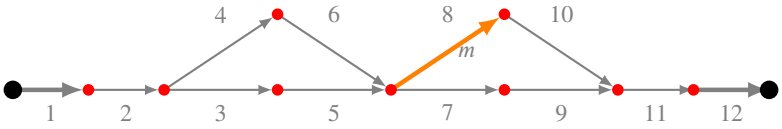

Figure 6: An example to explain constraints (11), (14) and (15): how to define sets $A' \subseteq A_m$ for a train m.

6. when we look at the physical network, if a long train impedes some switches at tick t, no other train can traverse that switch. The following constraints activate these switch incompatibilities between route arcs. Let $m \in M$ be a train that at tick t occupies route a with its head and does not fit within the stopping point of a. If at tick $t+1$ it is still with the head in a, we consider any subset $A' = \{a, a_1, \dots, a_k\}$ such that m impedes the switch at the end of a_k. I.e., being s the stopping node such that $a_k \in \delta^-(s)_m$, $l_m > c_a + \sum_{i=0}^{k-1} c_{a_i} + w_s$ (Note: the first of such sets to be considered will be $A' = \{a\}$ itself, and $a_k \equiv a$). Let $\bar{\mathcal{A}}'$ be the set of all such sets. Then no other train n enters a route arc which shares the switches with a_k at tick $t+1$:

$$x_{m,a,t} + \sum_{i=1}^{k} y_{m,a_i,t} + \sum_{i \in \mathcal{I}^{sw}(m,a_k)} (x_{n,i,t+1} - x_{n,i,t}) \leq k+1$$
$$\forall A' \in \bar{\mathcal{A}}', \forall m \in M, \forall n \in M, n \neq m, \forall t \in [T-1] \quad (14)$$

$$x_{m,a,t+1} + \sum_{i=1}^{k} y_{m,a_i,t+1} + \sum_{i \in \mathcal{I}^{sw}(m,a_k)} (x_{n,i,t+1} - x_{n,i,t}) \leq k+1$$
$$\forall A' \in \bar{\mathcal{A}}', \forall m \in M, \forall n \in M, n \neq m, \forall t \in [T-1] \quad (15)$$

7. no-swap constraints: a route arc occupied by a train at tick t cannot be occupied by a train in the opposite direction at tick $t+1$. Notice that this requirement is translated into: if at tick t train m occupies a route arc belonging to $\delta^-(s)_m$, at tick $t+1$ no train in the opposite direction can occupy a route arc incoming the same stopping node:

$$\sum_{m \in M_R} \sum_{a \in \delta^-(s)_m} (x_{m,a,t} + y_{m,a,t}) + \sum_{m' \in M_L} \sum_{a \in \delta^-(s)_{m'}} (x_{m',a,t+1} + y_{m',a,t+1}) \leq 1,$$
$$\forall s \in \bar{V}, \forall t \in [T-1] \quad (16)$$

$$\sum_{m \in M_R} \sum_{a \in \delta^-(s)_m} (x_{m,a,t+1} + y_{m,a,t+1}) + \sum_{m' \in M_L} \sum_{a \in \delta^-(s)_{m'}} (x_{m',a,t} + y_{m',a,t}) \leq 1,$$
$$\forall s \in \bar{V}, \forall t \in [T-1] \quad (17)$$

8. at the final tick, each train m must be placed at a route arc belonging to one of the following three subsets: (i) the black holes (B_m), (ii) the set of safe places (S_m), (iii) the initial position of train m. Constraints (19) ensure trains are stopped at their initial position whenever $h_m = 1$:

$$\sum_{a \in B_m} x_{m,a,T} + \sum_{a \in S_m} x_{m,a,T} + h_m = 1, \quad \forall m \in M \quad (18)$$

$$\sum_{a \in f(m, r_m^1)} x_{m,a,t} + h_m = 1 \quad \forall m \in M, \forall t \in [T] \quad (19)$$

The following sets of constraints describe the movements of dummy trains at ticks $[\bar{T}]$.

1. dummy trains originates at a black hole:

$$x_{\bar{m}, o_{\bar{m}}, T+1} = 1, \quad \forall \bar{m} \in \bar{M}, \quad (20)$$

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

2. at each dummy tick, each dummy train's head is allocated to exactly one of its feasible routes:

$$\sum_{a\in A} \quad x_{\bar{m},a,t} = 1, \quad \forall\, \bar{m} \in \bar{M},\, \forall\, t \in [\bar{T}] \tag{21}$$

3. at each dummy tick, a dummy train can only move forward (22), unless it already reached its final black hole (23):

$$x_{\bar{m},a,t} \quad \leq \sum_{a' \in f(a,\bar{m})} x_{\bar{m},a',t+1}, \quad \forall\, \bar{m} \in \bar{M},\, \forall\, a \in A \setminus \{o_{\bar{m}}\},\, \forall t \in [\bar{T}-1] \tag{22}$$

$$x_{\bar{m},d_{\bar{m}},t} \quad \leq x_{\bar{m},d_{\bar{m}},t+1}, \quad \forall\, \bar{m} \in \bar{M},\, \forall t \in [\bar{T}-1] \tag{23}$$

4. at each dummy tick, dummy trains cannot occupy route arcs incompatible with the ones occupied by trains in safe places (we recall that trains in safe place are still occupying the resources they occupied at tick T):

$$x_{\bar{m},a,t} \quad + \sum_{i\in I_a \cup a} (x_{m,i,T} + y_{m,i,T} - h_m) + \sum_{i\in I_a^{sw}:\, l_m > w_i} (x_m, i, T - h_m) \leq 1,$$
$$\forall \bar{m} \in \bar{M},\, \forall m \in M,\, \forall a \in A,\, \forall\, t \in [\bar{T}] \tag{24}$$

If a train m is stopped at its initial position, i.e. if $h_m = 1$, then these constraints are redundant, as both sums can only assume the values 0 or -1. This means that dummy trains can pass through m.

5. at tick $\bar{T}$ each dummy train has reached its destination:

$$x_{\bar{m},d_{\bar{m}},\bar{T}} = 1, \quad \forall \bar{m} \in \bar{M} \tag{25}$$

3.3. Objective function

The objective function is set to reflect the following expected behavior: the preferred outcome for a train is to be able to reach its black hole; if not, we try to find a safe place for it as close to its destination as possible; if this is also impossible, the train is stopped at its initial position. Hence, we define weights $u_{m,a}$ for each $m \in M$, $a \in A_m$, that increase with the distance from the destination (i.e., if arc $a' \in A_m$ is reachable from arc $a \in A_m$, then $u_{m,a} > u_{m,a'}$). We want to minimize the following objective function:

$$\min \quad \sum_{m\in M} \left(\sum_{a\in S_m} u_{m,a} x_{m,a,T} + U h_m \right) \tag{26}$$

where U is a constant big enough to make any feasible solution that does not stop any train at its initial position preferable to any solution with at least one $h_m = 1$.

Notice that, if all trains reach the black holes and none is bound-to-deadlock, the objective value is zero as the related variables do not appear in the objective function.

4. Validation of the approach

In this section we prove that our model can be used to solve the DD problem. In particular, we prove that if a deadlock is in action, then the Tick Formulation is able to detect it and that, otherwise, our model provides a feasible sequence of moves to route each train towards the extremes of the network. From these moves we can then produce a feasible schedule. Notice that, however, this schedule may be highly inefficient, as the time dimension can come out completely distorted. In fact, at this stage our primal goal is feasibility, and not computing a good-quality (e.g. with the lowest delays possible) schedule.

Proposition 1. A deadlock is in action if and only if the Tick Formulation detects a subset of bound-to-deadlock trains.

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

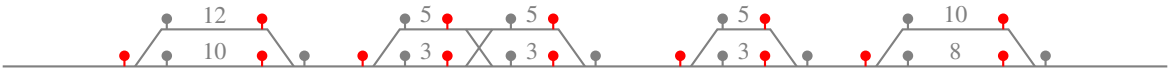

Figure 7: The railway network of Instance_8

Proof. As mentioned in the Introduction, the Tick Formulation reduces any reference to the time dimension to the definition of arbitrary ticks. To ensure that a deadlock-free solution found by our model can actually be translated into a feasible plan, we need to recover the time dimension and assign a schedule to each train. To construct such a schedule, we assume that, for each $m \in M$ and $a \in A_m$, we are given the time $\sigma_{m,a}$, necessary for train m to traverse route arc a.

Then, let $(\bar{x}, \bar{y})$ be a feasible solution to the Tick Formulation. For each train $m \in M$, let $(a_m^1, \dots, a_m^T)$ be the sequence of routes occupied by m at each tick t, according to the variables $\bar{x}_{m,a,t}$ that have value 1. We first assign a time instant to each tick. Therefore, for each $t \in [T]$, we let τ_t be the starting time of tick t and $M_t \subseteq M$ be the set of trains that actually move at t (those trains m such that $a_{m,t} \neq a_{m,t+1}$). We set τ_1 as the current time when the Deadlock Detector is triggered. Moreover, since tick $t+1$ can start only after each train $m \in M_t$ reached the stopping point $a_{m,t+1}$, we can use the following formula

$$\tau_{t+1} = \tau_t + \max_{m \in M_t} \sigma_{m,a_m^t} \tag{27}$$

to calculate the starting time of all ticks $t = 2, ..., T$. Then, at time τ_t, each train in $m \in M_t$ moves from $a_{m,t}$ to $a_{m,t+1}$.

On the contrary, whenever a feasible plan exists we can find a sequence of moves which routes the trains towards their destination. The easiest way of building a solution to the Tick formulation from a feasible plan is to assign sequentially a tick to each move of each train. □

Notice that this proof, hence the validity of the Tick Formulation to solve the DD problem, strongly rely on the assumption, introduced in Section 2, that trains can stop at each stopping node on their path.

5. Computational results

We performed some tests on a set of randomly generated instances that represent single-line scenarios. Hence, these instances consist of stations alternated to single tracks and are characterized by the following parameters: the number of stations, the number of platforms for each station and their length, the number of trains and their length. As deadlocks are most frequently caused by a small number of trains, we focused on instances with up to 7 trains.

Table 1 summarizes the characteristics of the instances tested. Column *Stations* indicates how many stations are found in the tested network. Column *Platforms* records how many platforms each station has and their length. For example, consider Instance_8 depicted in Figure 7: there are 4 stations, three of them are simple stations with two platforms and one, the second station, a more complex one with a total of 4 stopping points, two in a row. Column *Trains* records the length of the trains running on the network. Notice that the train-specific routes can be built starting from the railway network: in general, they will cover all the network routes ahead of the train's position, but routes may be left out if needed, for example to take into account compulsory stops. This is done before-hand, hence it does not figure in the computational times. We record on column *Deadlock* whether there are trains bound-to-deadlock or not and how many trains cause the deadlock (the number in brackets). Notice that we added here also very small instances (Instance_1 and Instance_2), made of three trains and only one station, to show that even very short trains may cause deadlocks.

The algorithm is implemented using Java language and instances are solved using branch and bound in Cplex 12.8, on a 2.5GHz Intel Core i5 processor with a 64-bit operating system with 8GB RAM. The timeout is set to 60 seconds, which already is a borderline threshold for real applications.

As mentioned in Section 2, it is not trivial to determine the minimum number of ticks necessary not to cut out a solution which succeeds at routing all trains to the black holes. However, the parameter T plays a relevant role in the performance of the algorithm. This is clearly seen in Table 2: for each instance we report the values of lower and upper bounds (resp. columns LB and UB) on the number of ticks. Then, we solved each instance several times varying the number of ticks. The first time it is solved with T equal to the lower bound on the number of ticks (column 0.0);

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

Table 1
Instances characteristics.

Instance	Stations	Platforms	Trains	Deadlock
Instance_1	1	(3,5)	1, 1, 1	No
Instance_2	1	(3,5)	1, 1, 8	Yes (3)
Instance_3	1	(3,5)	1, 4, 10	No
Instance_4	3	(3,5), (3,5), (9,11)	10, 15	No
Instance_5	3	(3,5), (3,5), (9,11)	5,10,15	No
Instance_6	3	(3,5), (3,5), (9,11)	10,10,15	Yes (3)
Instance_7	4	(5,7), (3,5), (7,9), (5,7)	18, 8, 8, 8, 18	Yes (4)
Instance_8	4	(10,12), (3,5)+(3,5), (3,5), (8,10)	15, 10, 5, 12, 5	No
Instance_9	4	(9,10), (3,5), (9,10), (5,7)	10, 10, 10, 5, 5, 5	Yes (3)
Instance_10	4	(9,10), (3,5), (9,10), (5,7)	10, 10, 5, 12, 5, 5, 10	Yes(4)
Instance_11	5	(8,10), (4,6)+(6,6), (8,10), (10,12), (6,8)+(6,8)	24, 24	Yes (2)
Instance_12	5	(8,10), (4,6)+(6,6), (8,10), (10,12), (6,8)+(6,8)	8, 18, 18, 30	Yes (4)
Instance_13	5	(8,10), (4,6)+(6,6), (8,10), (10,12), (6,8)+(6,8)	8, 18, 18, 18	Yes (4)
Instance_14	5	(8,10), (4,6)+(6,6), (8,10), (10,12), (6,8)+(6,8)	8, 8, 18, 18	No
Instance_15	5	(12,14), (5,7), (5,7), (5,7), (14,16)	10, 10, 5, 10, 8	Yes (3)
Instance_16	7	(10,12), (3,5), (3,5), (8,10), (3,5), (3,5), (12,14)	10, 10, 5, 5	No
Instance_17	7	(10,12), (3,5), (3,5), (8,10), (3,5), (3,5), (12,14)	10, 10, 12, 12	No
Instance_18	7	(10,12), (3,5), (3,5), (8,10), (3,5), (3,5), (12,14)	10, 15, 12, 12	Yes (3)
Instance_19	7	(10,12), (3,5), (3,5), (8,10), (3,5), (3,5), (12,14)	10, 8, 12, 12, 15	Yes (3)
Instance_20	7	(12,14), (5,7)+(5,7), (5,7), (10,12), (5,7), (5,7), (14,16)	20, 10, 20, 12, 5	Yes (3)

Table 2
Computational times in *ms* with different number of ticks.

Instance	LB	UB	0.0	0.2	0.4	0.6	0.8	1.0
Instance_1	4	8	1079	745	853	698	804	897
Instance_2	4	8	817	911	954	857	978	826
Instance_3	4	8	749	930	824	893	848	922
Instance_4	8	15	1095*	1035*	1202*	895	874	992
Instance_5	8	20	1150	1154	1040	1046	1307	926
Instance_6	8	20	1189	1024	1214	2254	2346	2777
Instance_7	10	34	1561	1715	2865	6558	16051	37309
Instance_8	11	33	1763	1537	1781	1407	1414	1429
Instance_9	10	37	2427	1788	6610	39852	24415	60000
Instance_10	10	42	1432	2237	4148	2459	3462	4232
Instance_11	14	27	1473	1524	3521	4584	10527	17602
Instance_12	12	39	1709	4605	60000	60000	60000	60000
Instance_13	12	39	1711	3559	60000	60000	60000	60000
Instance_14	12	39	1375	1727	1733	2080	2380	3273
Instance_15	12	42	1906	4789	9087	26224	60000	60000
Instance_16	16	50	5334*	2245	1141	1187	1869	2027
Instance_17	16	50	2940*	26490*	11409	43109	9881	5796
Instance_18	16	50	3230	11845	60000	60000	60000	60000
Instance_19	16	51	1867	60000	60000	60000	60000	60000
Instance_20	17	57	1864	13545	60000	60000	60000	60000

* Insufficient number of Ticks

then, we uniformly increase the number of ticks until we reach the upper bound (each column coefficient c represents a number of ticks equal to $LB+c(UB-LB)$). The numbers show the computational times in milliseconds. On one hand, values of T too near to the lower bound may induce *false positives*, which we do not want. In fact, trains are forced to stop before their black hole, but not because of a deadlock: there are not enough ticks available (see for example instance *Instance_3_(10,15)* in Table 2). On the other hand, values of T too near the upper bound consistently increase the computational time, especially with the increase of the size of the instance, making the approach impractical for real applications where fast decisions (to be executed in less than 10 seconds) are crucial.

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

6. Conclusions

In this paper we proposed a new pure 0, 1 integer formulation to solve the problem of detecting deadlocks. Moreover, as a way to provide a little help to the dispatchers towards the resolutions of deadlocks, and as a novelty with respect to other approaches, we are able to detect which trains need to stop on the spot and which ones still have a safe place they can reach. The Tick formulation is based on the idea that, left aside the time dimension, the trains movements can be decomposed into sequences of moves and we can find subsets of moves (at most one per each train) which can happen simultaneously. The maximum number of such subsets is equal to the sum of the number of moves for each train, but it is usually a redundant number that only increases the size of the formulation and, hence, the computational times.

Nevertheless, the formulation is polynomial with respect to the number of trains and the dimension of the network. It can also be easily customized, by adding extra sets of constraints to model peculiar circulation rules for a specific railroad manager. The main drawback of this formulation remains the difficulty of setting the right parameter for the number of ticks. For these reasons we are investigating improvements and alternative formulations for the deadlock detection problem.

CRediT authorship contribution statement

Veronica Dal Sasso: Methodology, Validation, Writing - Original Draft. **Leonardo Lamorgese:** Conceptualization, Methodology, Writing - Review & Editing. **Carlo Mannino:** Conceptualization, Methodology, Writing - Review & Editing, Supervision. **Andrea Onofri:** Software, Validation. **Paolo Ventura:** Conceptualization, Methodology, Writing - Review & Editing.

References

Ahuja, R., Magnanti, T., Orlin, J., 1993. Network flows: theory, algorithms and applications.

Arbib, C., Italiano, G.F., Panconesi, A., 1990. Predicting deadlock in store-and-forward networks. Networks 20, 861–881.

Aurizon, 2020. Second ertms pilot line commissioned. Railway Gazette International URL: https://www.railwaygazette.com/infrastructure/second-ertms-pilot-line-commissioned/55385.article. last Visited: 7 October, 2020.

Balas, E., 1985. On the facial structure of scheduling polyhedra. Mathematical Programming 24, 179–218.

Bollapragada, S., Markley, R., Morgan, H., Telatar, E., Wills, S., Samuels, M., Bieringer, J., Garbiras, M., Orrigo, G., Ehlers, F., et al., 2018. A novel movement planner system for dispatching trains. Interfaces 48, 57–69.

Caimi, G., Fuchsberger, M., Laumanns, M., Lüthi, M., 2012. A model predictive control approach for discrete-time rescheduling in complex central railway station areas. Computers & Operations Research 39, 2578–2593.

Caprara, A., Fischetti, M., Toth, P., 2002. Modeling and solving the train timetabling problem. Operations research 50, 851–861.

Cui, Y., 2009. Simulation based hybrid model for a partially automatic dispatching of railway operation. Ph.D. thesis. University of Stuttgart.

Cui, Y., Martin, U., Liang, J., 2017. Searching feasible resources to reduce false-positive situations for resolving deadlocks with the banker's algorithm in railway simulation. Journal of Rail Transport Planning & Management 7, 50–61.

Dyer, M., L., W., 1990. Formulating the single machine sequencing problem with release dates as a mixed integer program. Discrete Applied Mathematics 26, 255–270.

Fang, W., Yang, S., Yao, X., 2015. A survey on problem models and solution approaches to rescheduling in railway networks. IEEE Transactions on Intelligent Transportation Systems 16, 2997–3016.

Garey, M.R., Johnson, D.S., 1979. Computers and intractability. volume 174. freeman San Francisco.

Harrod, S., 2011. Modeling network transition constraints with hypergraphs. Transportation Science 45, 81–97.

Keha, A.B., Khowala, K., Fowler, J.W., 2009. Mixed integer programming formulations for single machine scheduling problems. Computers & Industrial Engineering 56, 357–367.

Koné, O., Artigues, C., Lopez, P., Mongeau, M., 2011. Event-based milp models for resource-constrained project scheduling problems. Computers & Operations Research 38(1), 3–13.

Li, F., Sheu, J.B., Gao, Z.Y., 2014. Deadlock analysis, prevention and train optimal travel mechanism in single-track railway system. Transportation research part B: methodological 68, 385–414.

Lu, Q., Dessouky, M., Leachman, R.C., 2004. Modeling train movements through complex rail networks. ACM Transactions on Modeling and Computer Simulation (TOMACS) 14, 48–75.

Mascis, A., Pacciarelli, D., 2002. Job-shop scheduling with blocking and no-wait constraints. European Journal of Operational Research 143, 498–517.

Mazzanti, F., Spagnolo, G.O., Della Longa, S., Ferrari, A., 2011. Deadlock avoidance in train scheduling: a model checking approach. International Workshop on Formal Methods for Industrial Critical Systems, Springer, Cham. , 109–123.

Meng, L., Zhou, X., 2014. Simultaneous train rerouting and rescheduling on an n-track network: A model reformulation with network-based cumulative flow variables. Transportation Research Part B: Methodological 67, 208–234.

Pachl, J., 2011. Deadlock avoidance in railroad operations simulation. Transportation Research Board 90th Annual Meeting Paper No. 11-0175, 359–369.

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239

Queyranne, M., Schulz, A.S., 1994. Polyhedral approaches to machine scheduling. Berlin: TU, Fachbereich 3 .

Reynolds, E., Ehrgott, M., Maher, S.J., Patman, A., Wang, J.Y., 2020. A multicommodity flow model for rerouting and retiming trains in real-time to reduce reactionary delay in complex station areas. submitted URL: http://www.optimization-online.org/DB_FILE/2020/05/7816.pdf. last visited: 10.10.2020.

Tesch, A., 2020. A polyhedral study of event-based models for the resource-constrained project scheduling problem. Journal of Scheduling , 1–19.

Wen, C., Huang, P., Li, Z., Lessan, J., Fu, L., Jiang, C., Xu, X., 2019. Train dispatching management with data-driven approaches: a comprehensive review and appraisal. IEEE Access 7, 114547–114571.

This is the author accepted manuscript/postprint version of an article published in Journal of Rail Transport Planning & Management.
https://doi.org/10.1016/j.jrtpm.2021.100239